Dr. Mindaugas Šarpis

Lessons on **Data Analysis** from **CERN**

Index of Lectures

Index of Lectures

Dr. Mindaugas Šarpis

Lessons on **Data Analysis** from **CERN**

Lecture 1:
Course Orientation and Motivation

Course Structure

- Lectures
  - Theory / Overviews
    - Main goal is exposure
  - Discussion
    - Building intuition. Interactivity is important.
- Seminars
  - Demos
  - Hands-on Sessions
    - *Inverted Classroom*
  - Case Studies

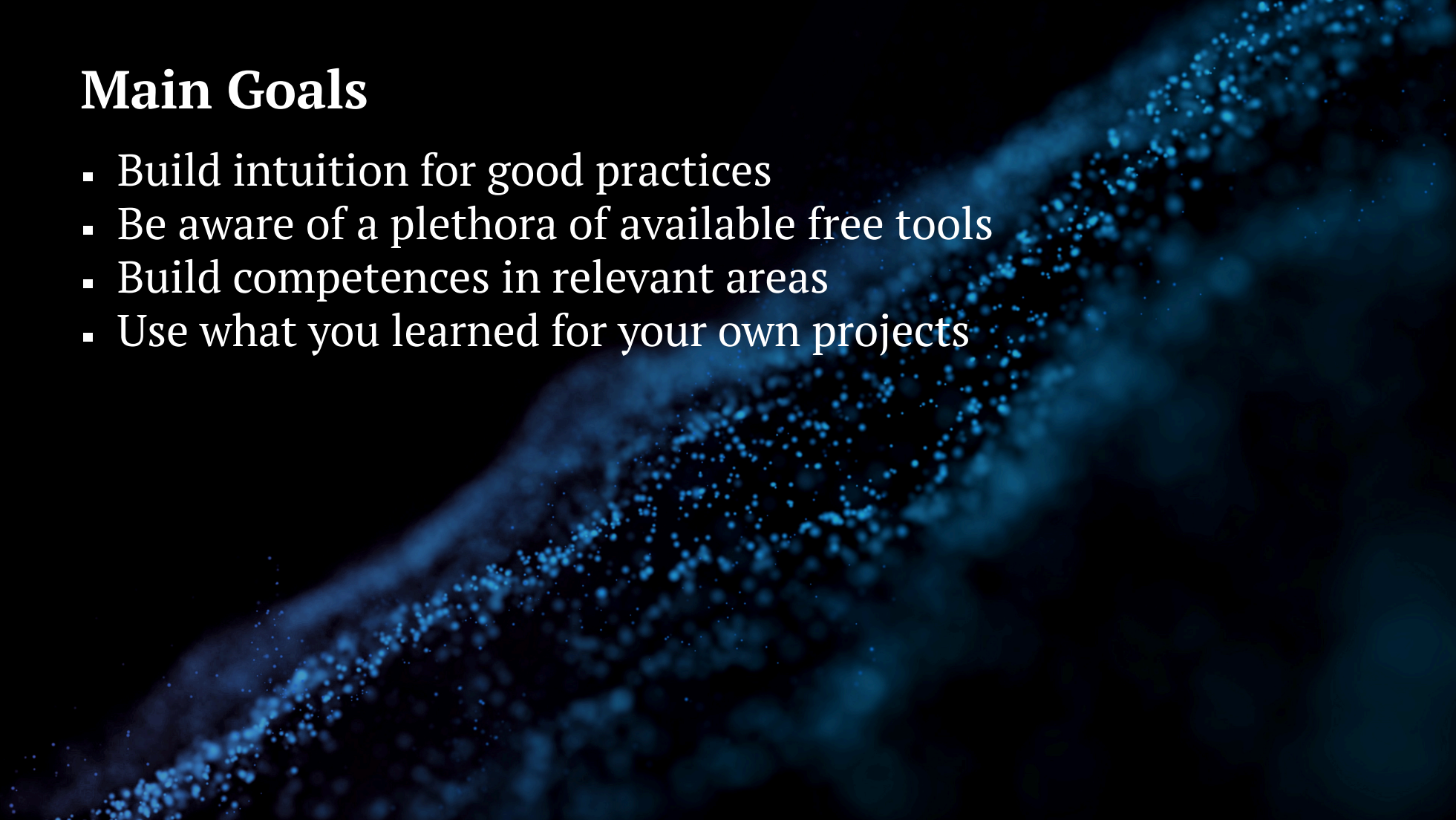
Grading Structure

- **2 x 20%** Quiz
- **1 x 60%** Final Project

* One quiz from data analysis part and one from AI part
** Students are graded on the project according to their previous expertise
One should be able to explain what they did, how, and why

Main Goals

- Build intuition for good practices
- Be aware of a plethora of available free tools
- Build competences in relevant areas
- Use what you learned for your own projects



Project Details

- Should include elements of data analysis, using good practices and tools
- May be related to your field of study/current work
- Can focus on a specific tool/method/algorithm or a broader workflow
- Codebase made available on GitHub
- Project written up in a one-page report
- Final presentation at the end of the course is graded on the spot

Lecture Plan for the Data Analysis Part

February 6 (2 lectures)	March 20 (2 lectures)
February 7 (2 lectures)	March 21 (2 lectures)
February 8 (4 lectures)	March 22 (4 lectures)

Course content (prone to change)

- Common pitfalls in working with computers
- File handling and directory structure
- Concepts of Data Analysis
- Main principles of computing
- Computing infrastructure
- Crash course on Python programming
- Different Computing Environments
- Creating robust workflows
- Crash course on statistics
- Ideas for projects

Goals of the course

After this course you should:

- Understand main concepts of computing
- Know which tools to chose for a specific task
- Be able to implement simple data analysis workflows on the fly
- Be safe from common pitfalls in working with computer
- Gain knowledge on mathematics and statistics needed for data analysis
- Understand the basics of machine learning and AI
- Become platform and tool agnostic in your work
- Be able to adapt to new tools and technologies quicker


Who am I talking to?

Dr. Mindaugas Šarpis

Lessons on **Data Analysis** from **CERN**

Lecture 2:
Intro to CERN

No slides provided for this lecture. If you are interested in CERN, please visit CERN's website. There are many resources available for you to explore.

Dr. Mindaugas Šarpis

Lessons on **Data Analysis** from **CERN**

Lecture 3:
File Handling and Directory Structure


Common pitfalls in working with computers

File management chaos

Common issues:

- "I have no idea where I saved that file"
- "My file is gone!"
- "I have 10 files with the same name, which one is the right one?"
  - `final_final_v2.docx`, `asdfasdf.docx`, `asdfasdf.docx`
- "I have overwritten my file with the wrong version"

How to avoid:

- Create a consistent folder structure
- Use descriptive filenames and version numbers
- Employ file tagging, search filters, or integrated version control systems like Git to help keep track of changes

No backups

- "I lost all my data"
- "I accidentally deleted my file"
- "My computer crashed and I lost everything"
- "I spilled tea on my laptop now my thesis is gone"

How to avoid:

- Use automatic cloud backup services (Dropbox, Google Drive, OneDrive).
- Keep external backups on physical drives, ensuring they’re in a separate location.
- Consider version control for text-based files (Git), so you can revert to an older version if needed.

Compatibility issues

- "I can’t open this file"
- "This only works on my old laptop"
- "I have a mac so this probably won’t work"
- "I opened this word file but it’s all broken"
- "The script was running ok but now I get errors"

How to avoid:

- Use open-source software and file formats whenever possible
- Use cloud-based tools that work across different platforms
- Use virtual machines or containers to ensure compatibility
- Use version control to track changes and revert to a working version
- Use software that is actively maintained and updated
- Use software that is widely used and has a large user base
- Agree on software and file formats with collaborators

File Naming Conventions

- **Think about your files beforehand**
  - Identify what group of files your naming convention will cover
  - You can use different conventions for different file sets
  - Check for established file naming conventions in your discipline or group

- **Identify metadata**
  - Experiment conditions
  - Type of data
  - Researcher name/initials
  - Lab name/location
  - Project or experiment name or acronym
  - Date or date range of experiment
  - Experiment number or sample ID
- **Abbreviate and encode metadata in the file name**
  - Decide what shortened information to keep
  - Standardize the categories and/or replace them with 2- or 3-letter codes
  - Be sure to document these codes

- **Use Versioning**
  - Use versioning to indicate the most current version of a file
  - Track versions of a file by adding version information to end of the file name, e.g. filename_v2.xxx
  - Use a version number (e.g. “v01” or “v02”)
  - Use the version date (use ISO 8601 format: YYYYMMDD or YYYY-MM-DD)

- **Ensure Files are Searchable**
  - Think about how you want to sort and search for your files in order to determine the order for the metadata in the file name
  - Decide what metadata should appear at the beginning
  - Use default ordering: alphabetically, numerically, or chronologically
  - Use ISO 8601-formatted dates (YYYYMMDD or YYYY-MM-DD)

- **Separate Metadata Elements**
  - Use dashes (-), underscores (_), or capitalize the first letter of each word
    - Dashes: file-name.xxx
    - Underscores: file_name.xxx
    - No separation: filename.xxx
  - Camel case (the first letter of each section of text is capitalized): FileName.xxx
  - Avoid special characters, such as: ~ ! @ # $ % ^ & * ( ) ` ; : < > ? . , [ ] { } ’ " |

- **Write down your naming conventions**
  - If the file is moved or shared, users will be able to identify the file from its file name
  - File names should be 40-50 characters and conventions should only use alphanumeric characters, dashes, underscores
  - If you find that you are encoding a large amount of metadata in the file names, you should consider storing this metadata in a master spreadsheet with your data for future reference

Two different file types

- **Text File**
  - Human readable
  - Can be opened with any text editor
  - Generally larger
  - Usually config files, logs, or scripts
- **Binary File**
  - Not human readable
  - Requires specific software to open
  - Generally smaller
  - Usually images, videos, or executables


Demo
Files and Extensions

File Organization: Naming

Overview

Teaching: 30 min
Exercises: 10 min
Questions

- What are the common file organization errors?
- What are best practices for file organization?

Objectives

- Highlight common SNAFUs
- Learn to employ unit testing

Names matter

Directory Structure

Organized by File type

```
|- Data/
|  |- Processed/
|  |- Raw/
|- Results/
|  |- Figure1.tif
|  |- Figure2.tif
|  |- Models/
|  |  |- Model1/
```

Organized by Analysis

```
|- Figure1/
|  |- Data/
|  |- Results
|  |  |- Figure1.tif
|- Figure2/
|  |- Data/
|  |- Results/
|  |  |- Figure2.tif
```

Dr. Mindaugas Šarpis

Lessons on **Data Analysis** from **CERN**

Lecture 4
Concepts of Data Analysis

What is Data Analysis?

What is **Data**?

interactive exercise


Data analysis is a process of inspecting, cleaning, transforming, and modeling **data** with the goal of discovering useful **information**, informing conclusions, and supporting decision-making

Wikipedia


What is Data Science?


Data science is an interdisciplinary academic field that uses statistics, scientific computing, scientific methods, processing, scientific visualization, algorithms and systems to extract or extrapolate **knowledge and insights** from potentially noisy, structured, or unstructured data
Wikipedia

- **Key Ideas**
  - Any experiment (study or analysis) in any field of science will have a data analysis component
  - Normally, the **results of data analysis** appear in scientific **publications**
  - In business data analysis is imperative for **decision making**


Examples of **data analysis** in different fields of science and industry

Bio medicine and Genomics

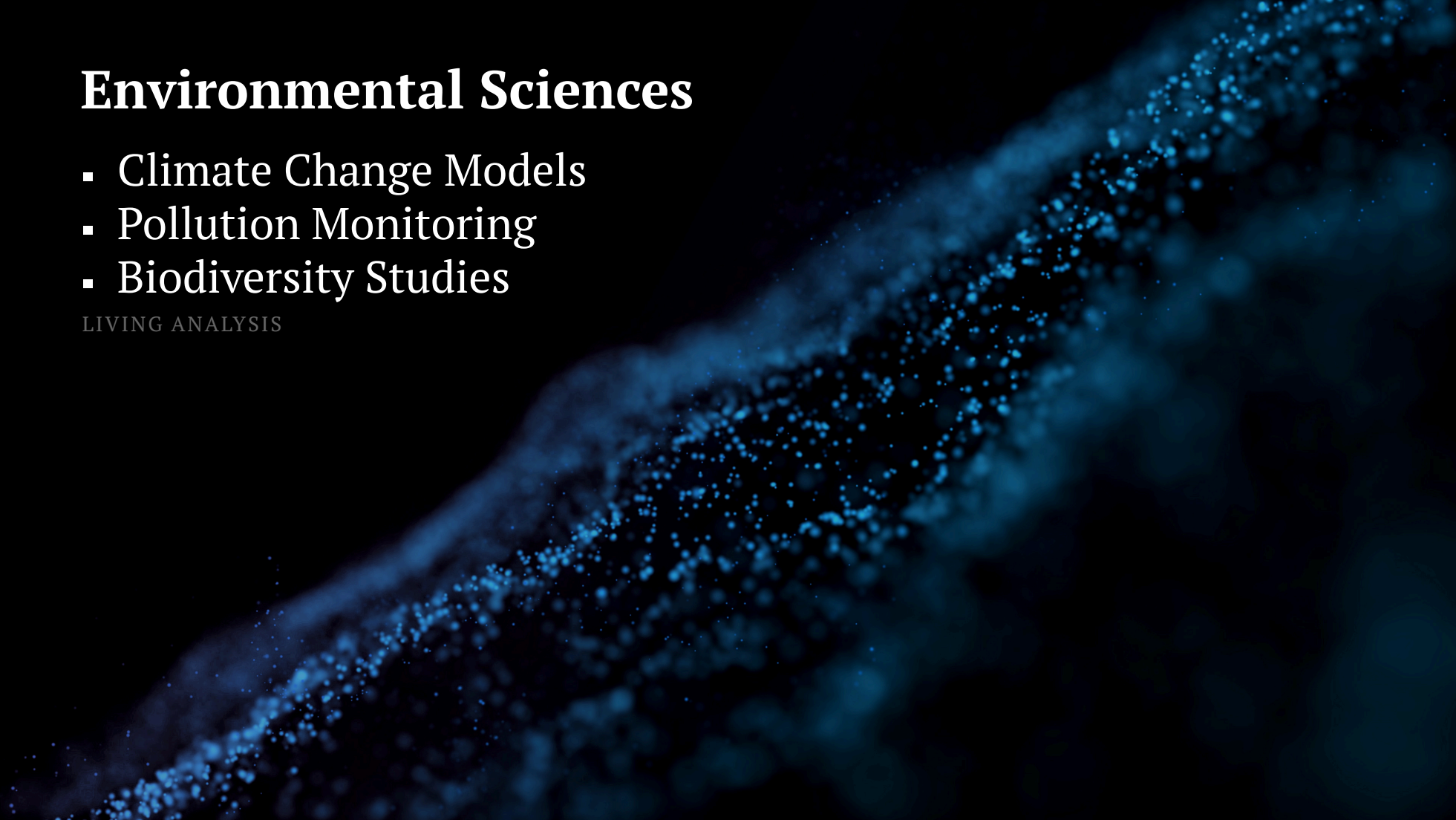
- Genome Sequencing
- Clinical Trials

23andMe anyone (ancestry services)?
Comparing against *reference populations*

Environmental Sciences

- Climate Change Models
- Pollution Monitoring
- Biodiversity Studies

LIVING ANALYSIS



Social Sciences

- Economic Forecasting
- Social Behavior Studies

MAY BE QUALITATIVE ANALYSIS

Astronomy

- Observational Data Analysis
- Gravitational Waves
- …

Engineering

- Predictive Maintenance
- Quality Control
- Structural Health Monitoring

Healthcare

- Epidemiology
- Health Policy
- ...

Finance

- Stock Market Analysis
- Risk Management
- Algorithmic Trading

Sports Analytics

- Performance Analysis
- Fan Engagement

Steps of Data Analysis

- 1. **Define the Problem or Research Question**
  - Formulation

    This might steer the choices in the following steps
  - Experimental Design

INTERACTIVE EXERCISE

- 2. **Collect Data**
  - How much data do you need?
  - What sort of data do you need?
  - What data formats should you chose?
  - Can you trust the data?
  - Can you collect the data?

- 3. **Clean Data**
  - Data Selection
  - Data Stripping
  - Data Skimming
  - Data Wrangling
  - …

- 4. **Analyze Data**
  - Data Exploration
  - Statistical Analysis
  - Model Building
  - Machine Learning
  - Classification (…**AI**…)

- 5. **Visualize the data**
  - What’s your target audience?
  - What is the message you want to convey?

- 6. **Interpret and report the results**
  - Draw Conclusions from Data
  - Report Findings


Data Hygiene


FAIR

The first step in **(re)using data** is to find them. **Metadata** and data should be easy to find for both humans and computers. Machine-readable metadata are essential for automatic discovery of datasets and services, so this is an essential component of the FAIRification process.

Findable data

- **F1.** (Meta)data are assigned a globally **unique** and persistent **identifier**
- **F2.** Data are described with **rich metadata**
- **F3. Metadata** clearly and explicitly **include the identifier** of the data they describe
- **F4.** (Meta)data are registered or indexed in a **searchable resource**

Accessible data

- **A1. (Meta)data** are retrievable by their **identifier** using a standardised communications protocol
  - **A1.1** The protocol is **open**, free, and universally implementable
  - **A1.2** The protocol allows for an **authentication** and **authorisation** procedure, where necessary
- **A2.** Metadata are accessible, even when the data are no longer available

Interoperable data

- **I1.** (Meta)data use a formal, accessible, shared, and broadly applicable **language for knowledge representation**
- **I2.** (Meta)data use vocabularies that follow **FAIR principles**
- **I3.** (Meta)data include **qualified references** to other (meta)data

Reusable data

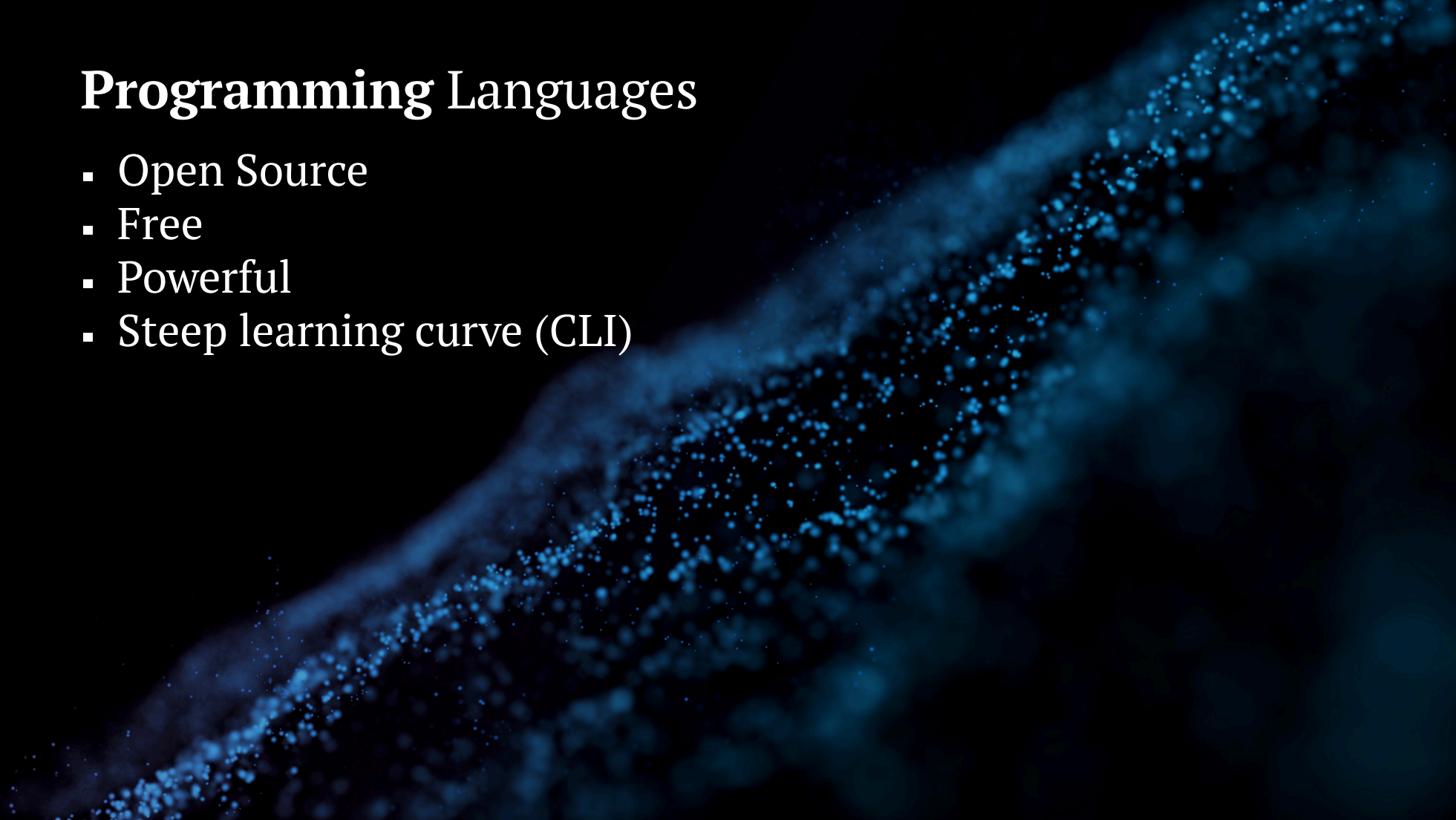
- **R1.** (Meta)data are **richly described** with a plurality of accurate and relevant attributes
  - **R1.1.** (Meta)data are released with a clear and **accessible** data usage **license**
  - **R1.2.** (Meta)data are associated with detailed **provenance**
  - **R1.3.** (Meta)data meet **domain-relevant community standards**

Proprietary Tools

Programming Languages

Proprietary Tools

- Expensive
- Limited in scope
- Lack compatibility
- Lack flexibility
- Easy to learn / use (GUI)



Programming Languages

- Open Source
- Free
- Powerful
- Steep learning curve (CLI)

Discussion

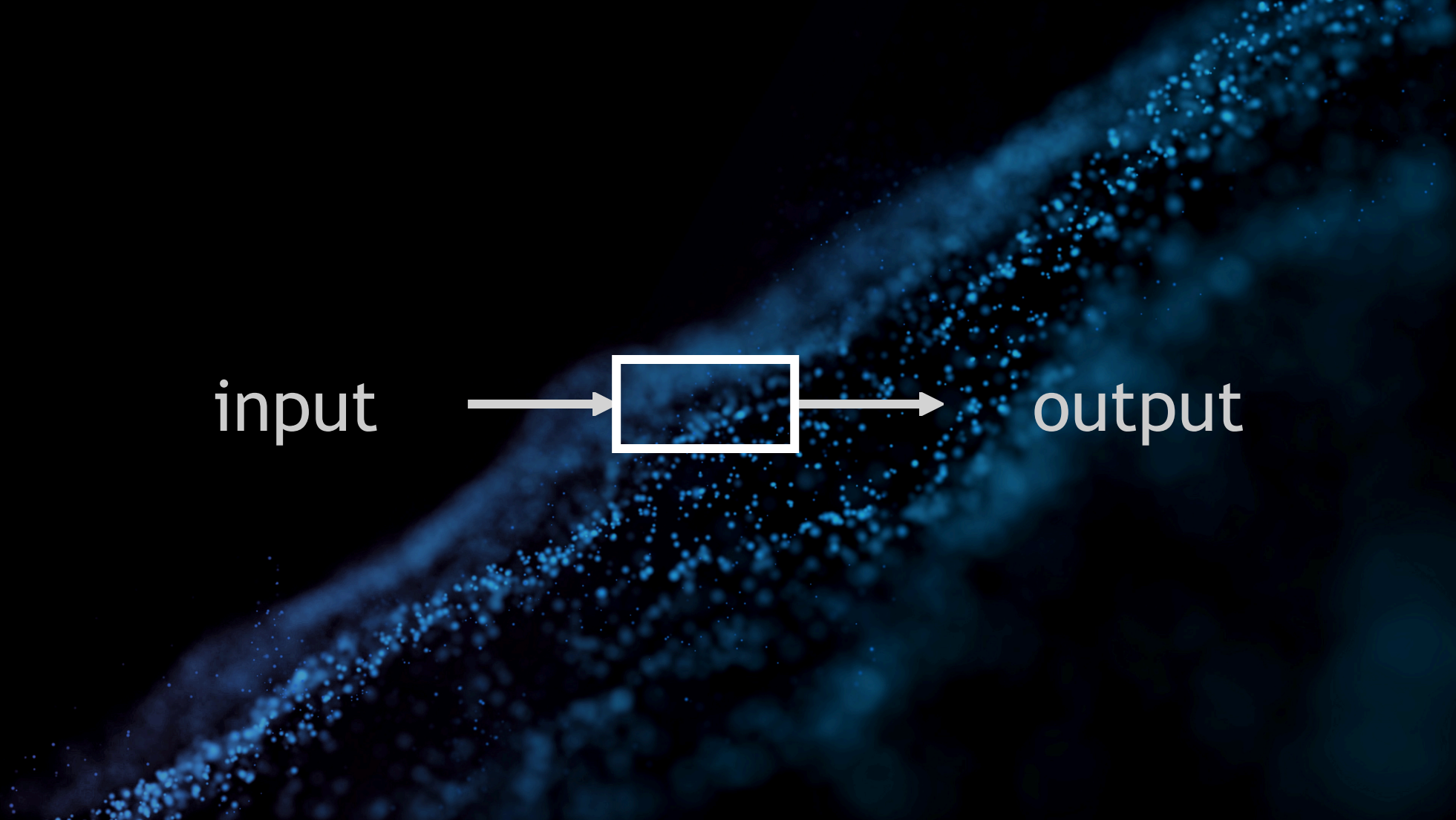
- When to use proprietary tools?
- What should you be using?
- Saturation of achieved proficiency

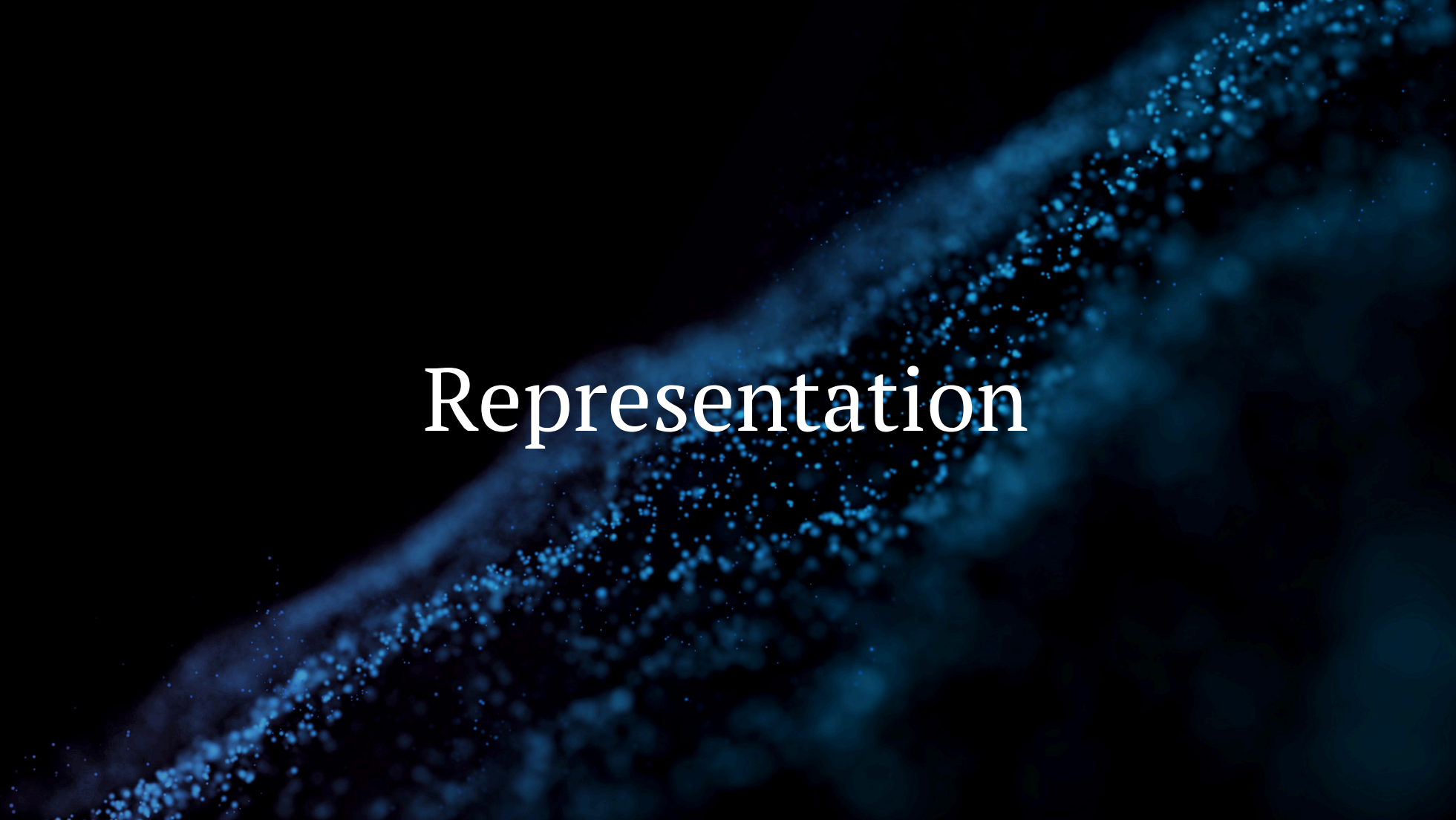
Dr. Mindaugas Šarpis

Lessons on **Data Analysis** from **CERN**

Lecture 5:
Crash Course on Computer Science


The main goal of this lecture is to promote algorithmic thinking and to provide a basic understanding of computer science concepts.


input
output


Representation


Unary
Base-1


Binary
Base-2

0

1


Binary Digit


Bi
t


Bit

Decimal

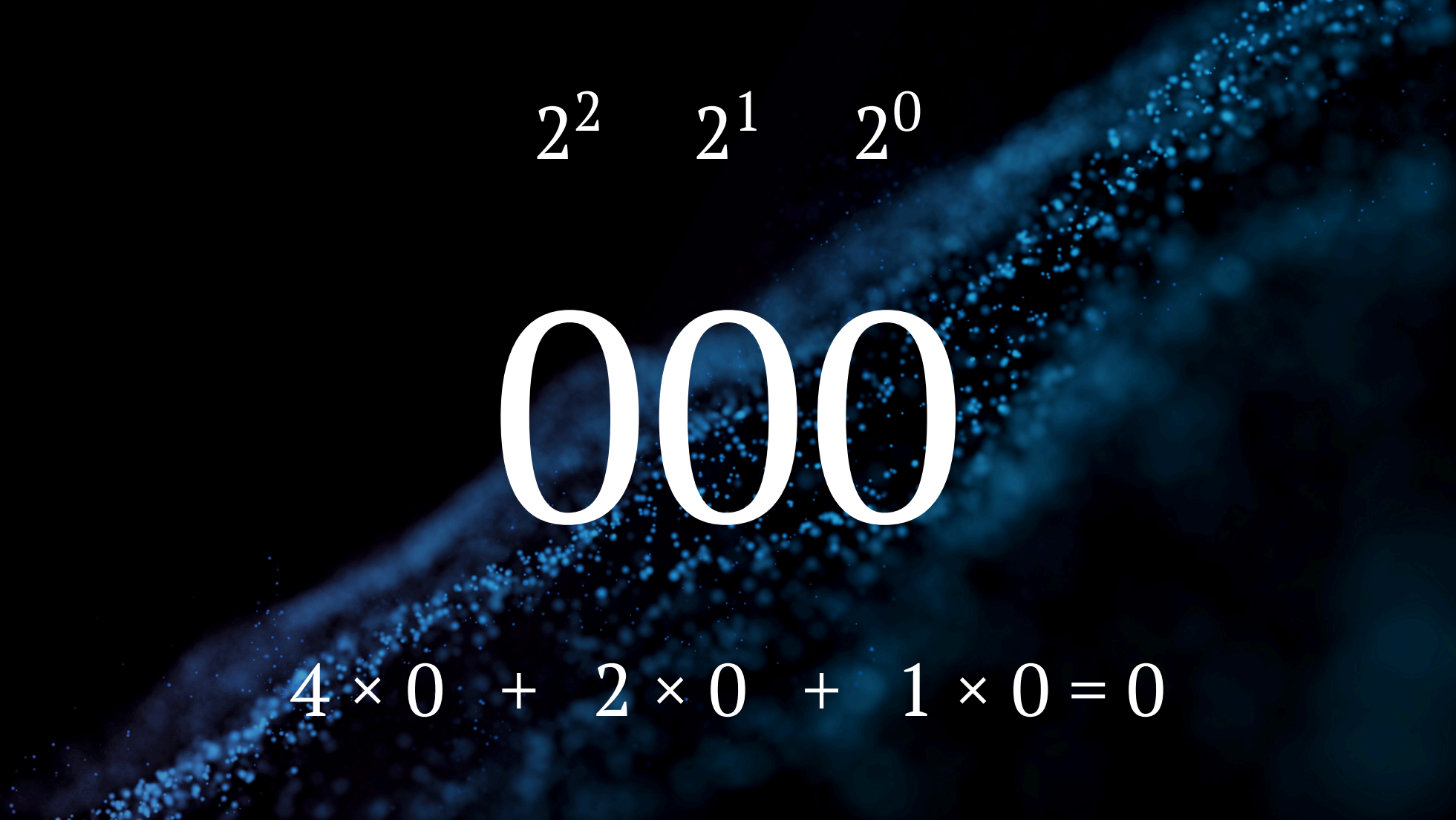
Base-10

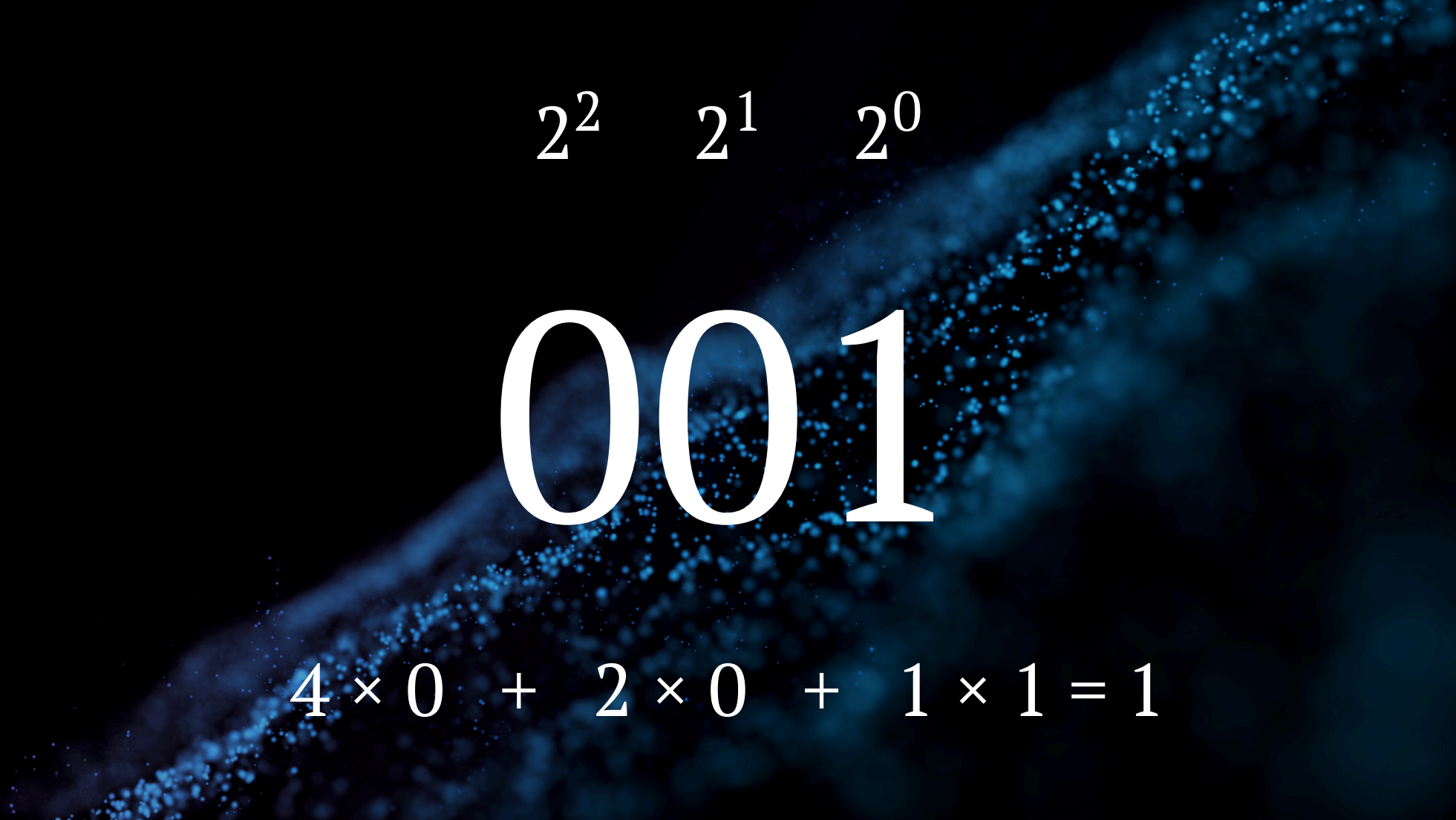

100 10 1
123
100 × 1 + 10 × 2 + 1 × 3

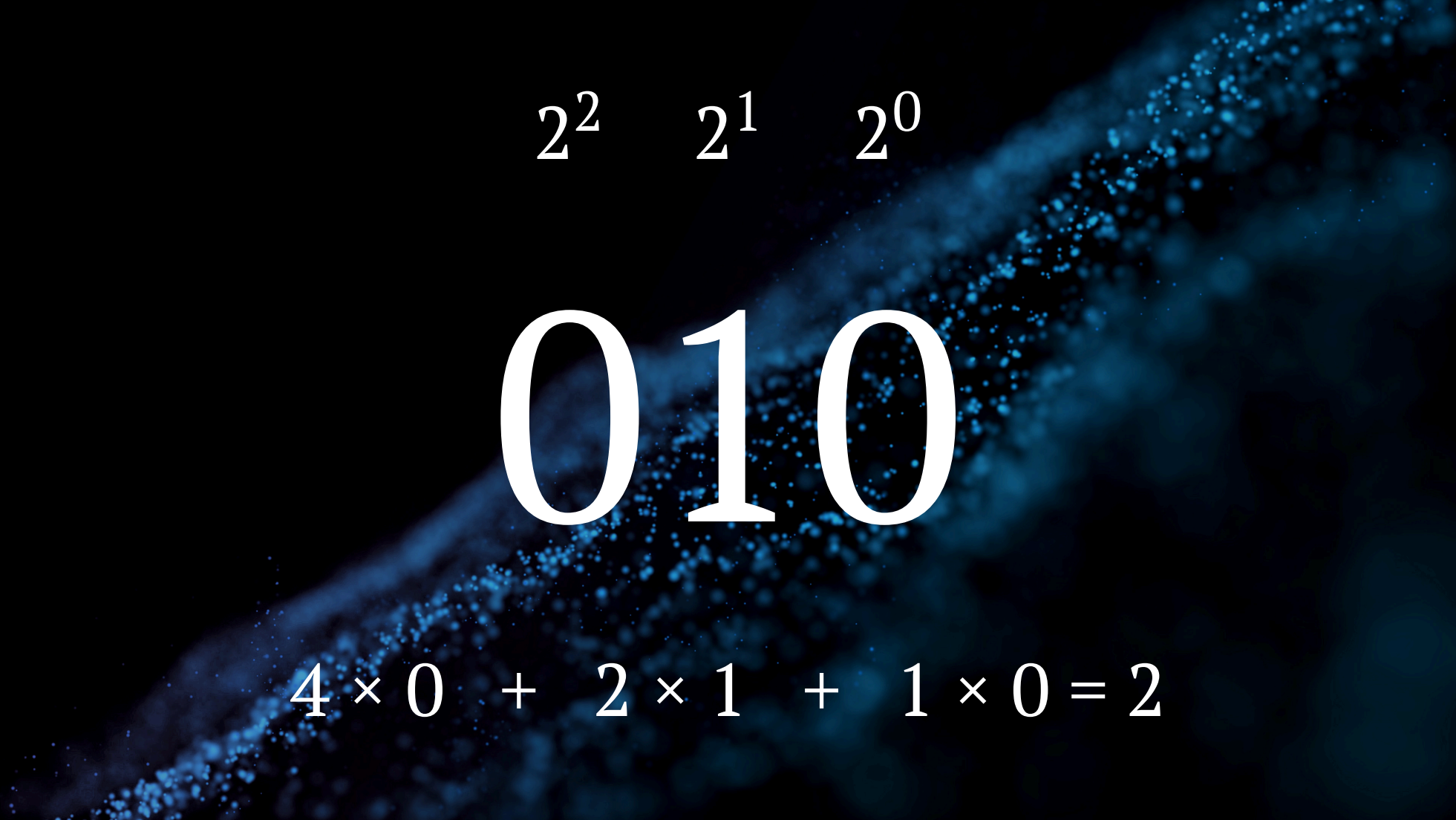
10^2 10^1 10^0

123

$100 \times 1 + 10 \times 2 + 1 \times 3$


2^2 2^1 2^0
000
$4 \times 0 + 2 \times 0 + 1 \times 0 = 0$


2^2 2^1 2^0
001
$4 \times 0 + 2 \times 0 + 1 \times 1 = 1$


2^2 2^1 2^0
010
$4 \times 0 + 2 \times 1 + 1 \times 0 = 2$

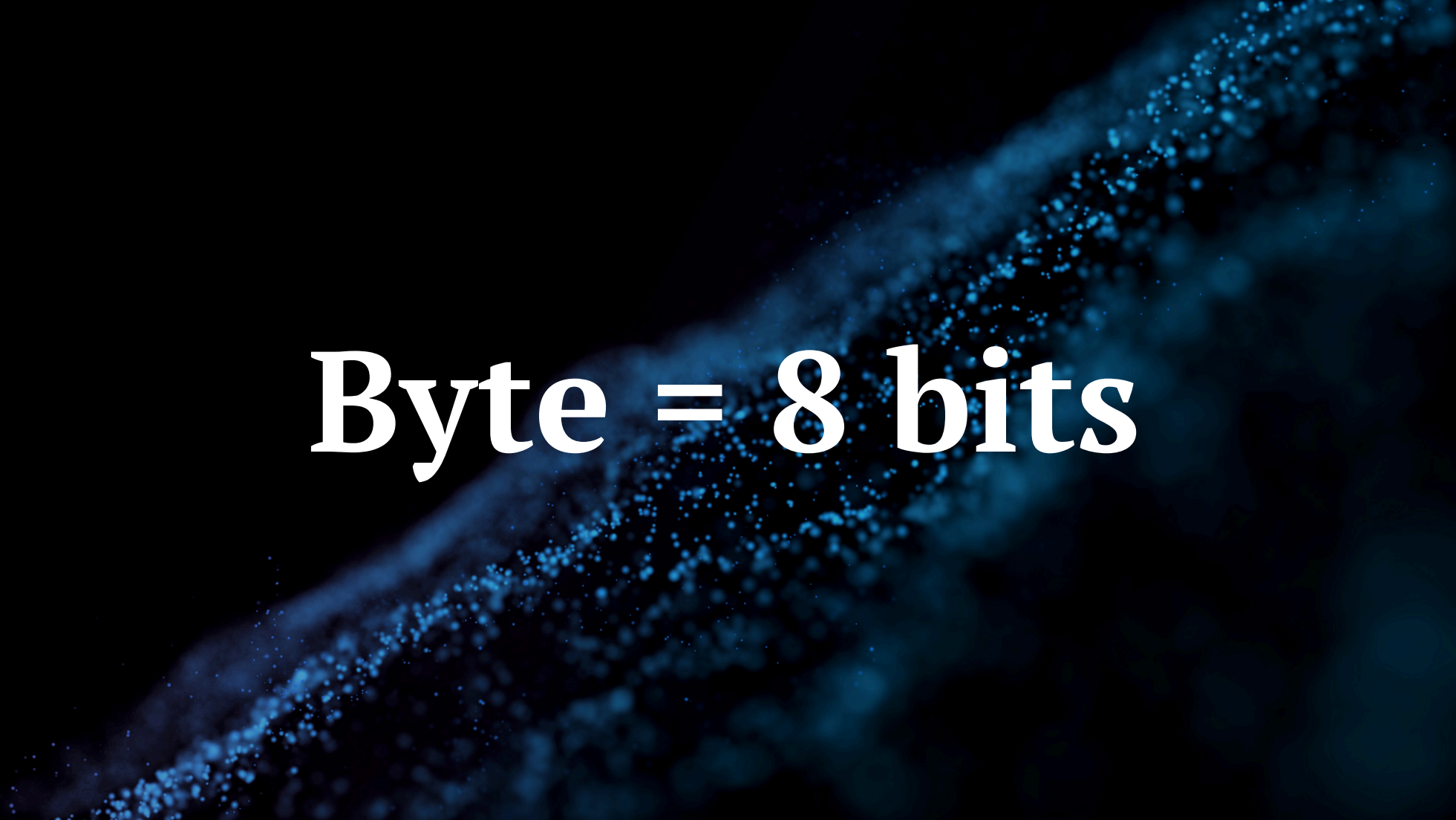
$$2^2 \quad 2^1 \quad 2^0$$

011

$$4 \times 0 \; + \; 2 \times 1 \; + \; 1 \times 1 = 3$$


2^2 2^1 2^0
100
$4 \times 1 + 2 \times 0 + 1 \times 0 = 4$


Byte


Byte = 8 bits


00000000


11111111


ASCII

0	**NUL**	16	**DLE**	32	**SP**	48	**0**	64	**@**	80	**P**	96	**`**	112	**p**
1	**SOH**	17	**DC1**	33	**!**	49	**1**	65	**A**	81	**Q**	97	**a**	113	**q**
2	**STX**	18	**DC2**	34	**"**	50	**2**	66	**B**	82	**R**	98	**b**	114	**r**
3	**ETX**	19	**DC3**	35	**#**	51	**3**	67	**C**	83	**S**	99	**c**	115	**s**
4	**EOT**	20	**DC4**	36	**$**	52	**4**	68	**D**	84	**T**	100	**d**	116	**t**
5	**ENQ**	21	**NAK**	37	**%**	53	**5**	69	**E**	85	**U**	101	**e**	117	**u**
6	**ACK**	22	**SYN**	38	**&**	54	**6**	70	**F**	86	**V**	102	**f**	118	**v**
7	**BEL**	23	**ETB**	39	**’**	55	**7**	71	**G**	87	**W**	103	**g**	119	**w**


If you are interested in learning more basics of computer science, a great resource is an open course by Harward University called **CS50**

Dr. Mindaugas Šarpis

Lessons on **Data Analysis** from **CERN**

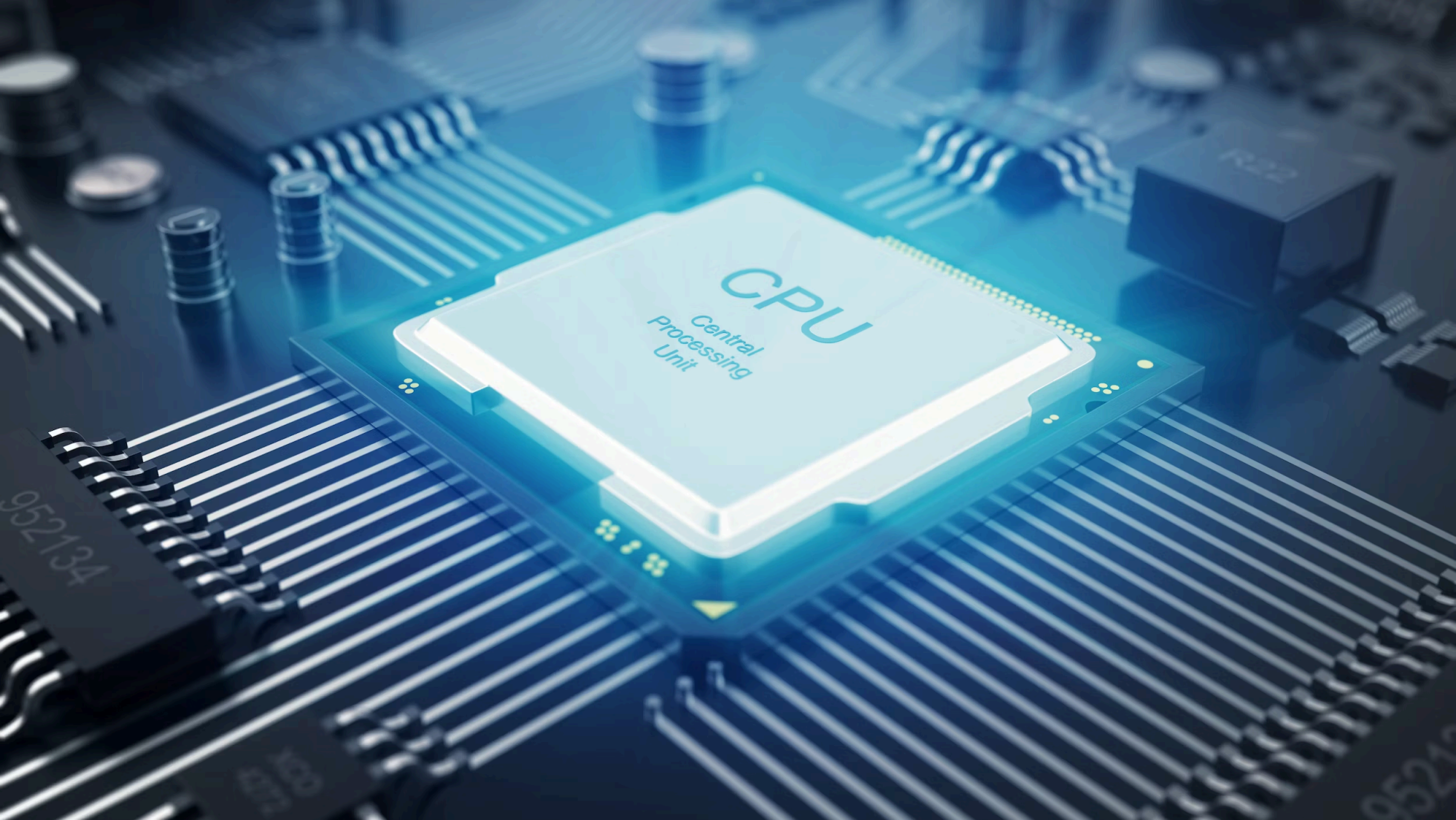
Lecture 6:
Computing Infrastructure

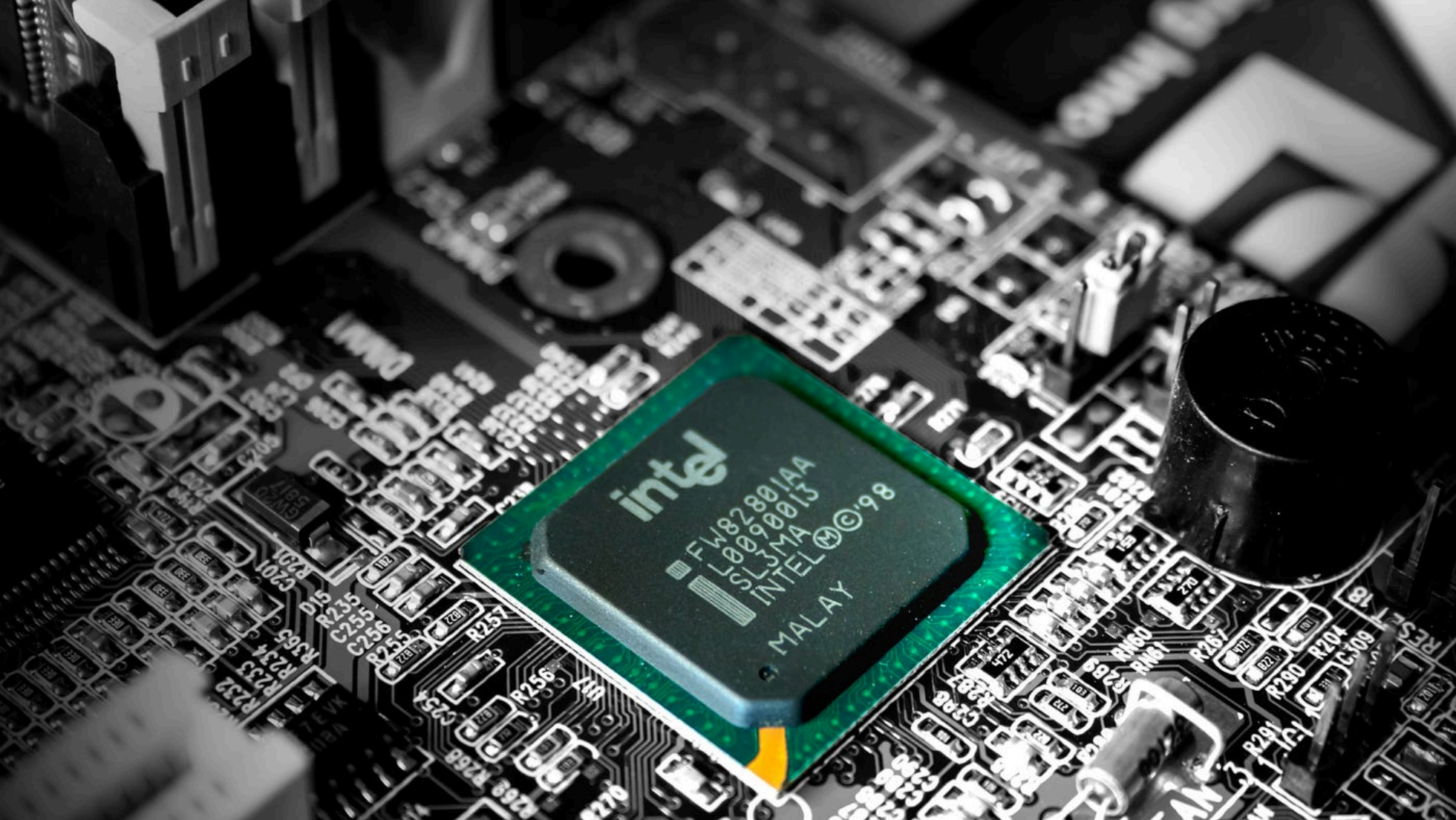
What constitutes computing infrastructure?

- **Hardware Components**
  - Central Processing Unit (**CPU**)
  - Memory (**RAM**)
  - Storage Devices (**HDD, SSD, NVMe**)
  - Input/Output (**I/O**) Devices
  - Specialized Processors (**GPUs**, **TPUs**)

- **CPU** (Central Processing Unit)
  - Basic arithmetic, logic, control, and input/output operations
  - CPU sub-components
    - Control Unit (CU)
    - Arithmetic Logic Unit (ALU)
    - Registers
    - Cache
    - Buses

- **CPU Performance** Factors:
  - Clock Speed
  - Number of Cores
  - Cache Size
  - Power Efficiency


CPU
Central
Processing
Unit


intel
FW82801AA
L009013
SL3MA
INTEL ©'98
MALAY

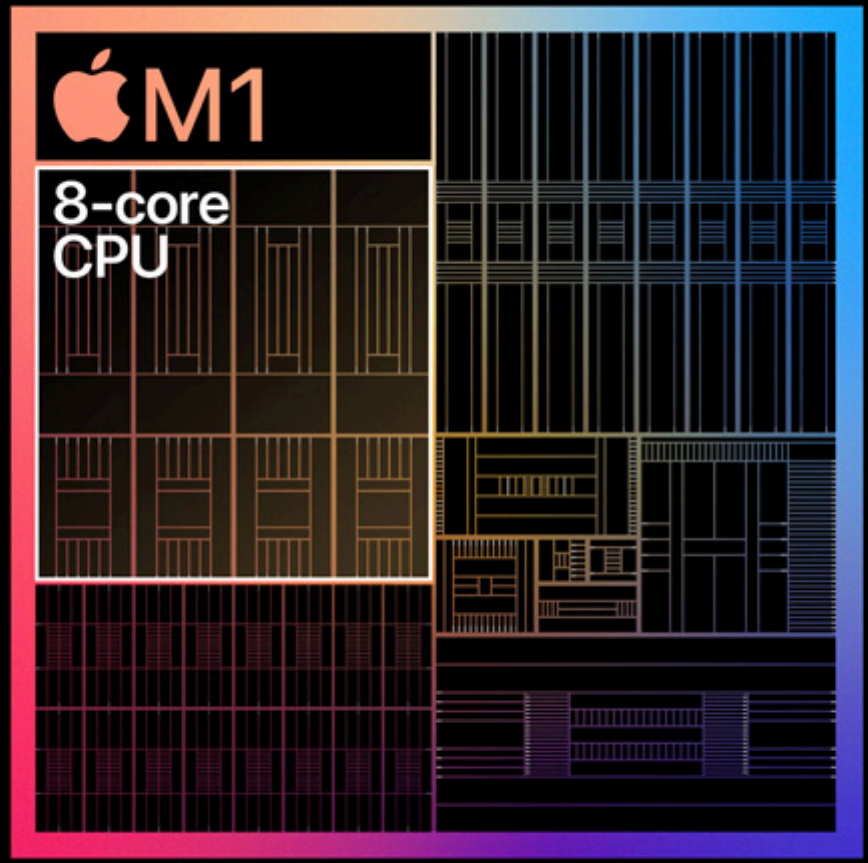
8-core CPU

The highest-performance CPU we've ever built.

Up to

3.5x

faster CPU performance[1]



- **RAM** (Random Access Memory)
  - Volatile memory
  - High-Speed Access
  - Temporary Storage
  - Capacity (GB or TB)
  - Performance (MHz or GHz)


CORSAIR
DOMINATOR
DOMINATOR
DOMINATOR
DOMINATOR
CORSAIR
PUMP_FAN1

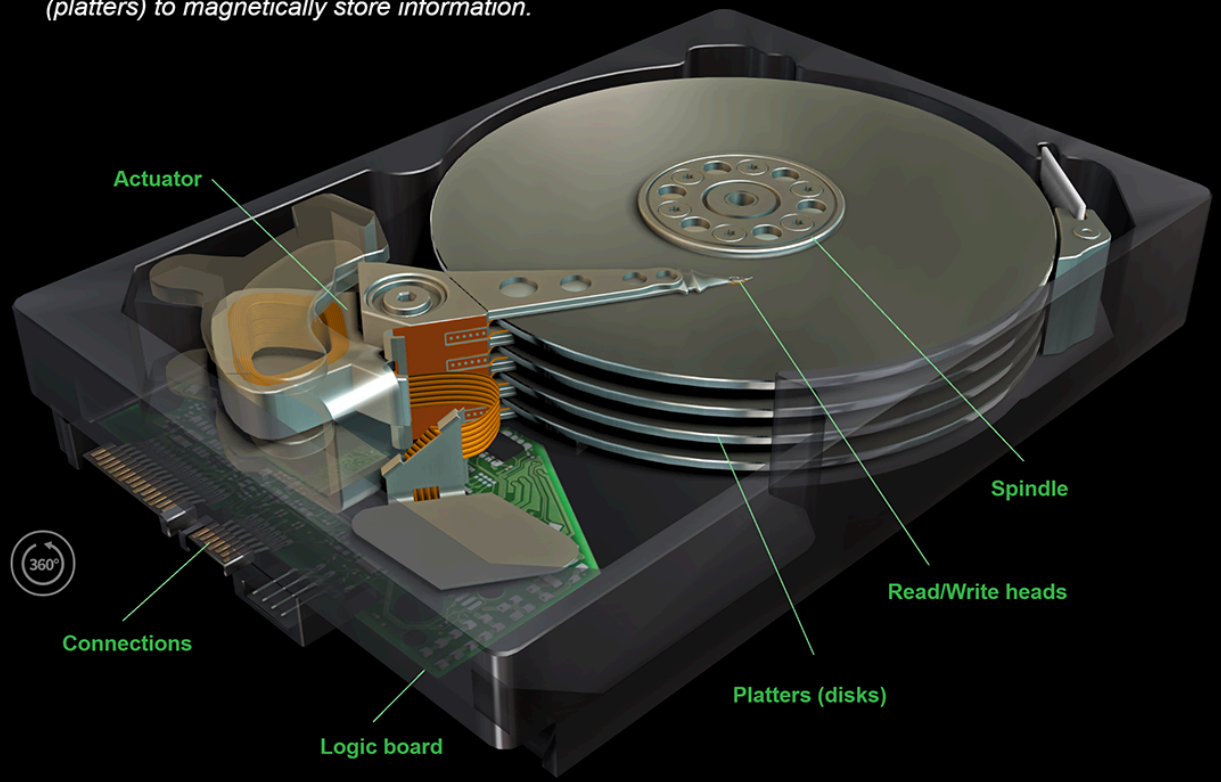
- **Storage** Devices:
  - **HDD** (Hard Disk Drive)
  - **SSD** (Solid State Drive)
  - **SSHD** (Solid State Hybrid Drive)
  - **NVMe** (Non-Volatile Memory Express)

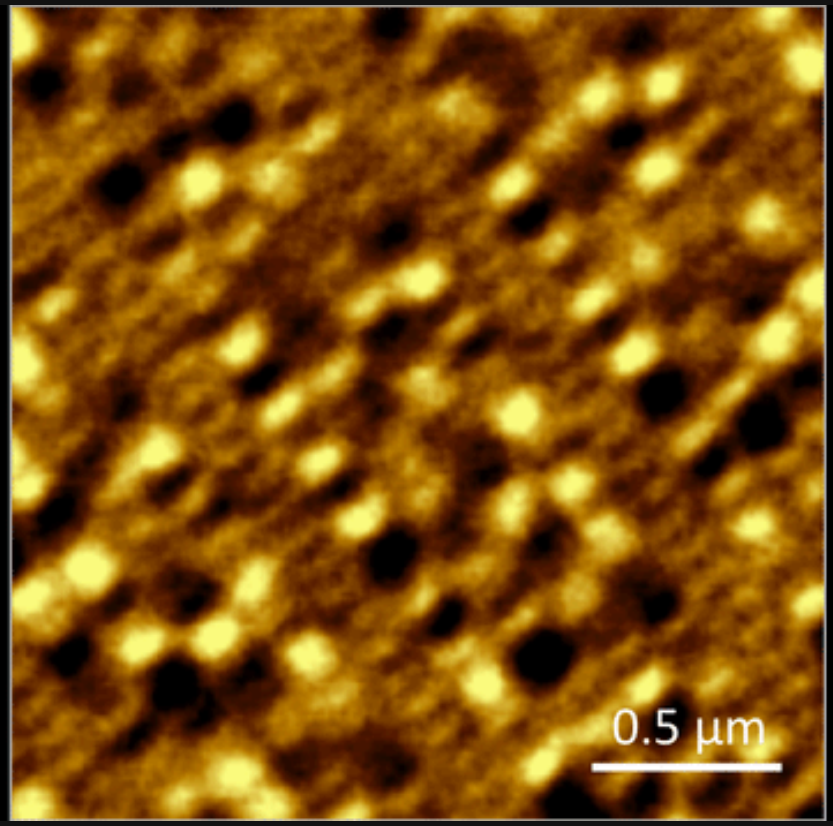
How Hard Disk Drives Work

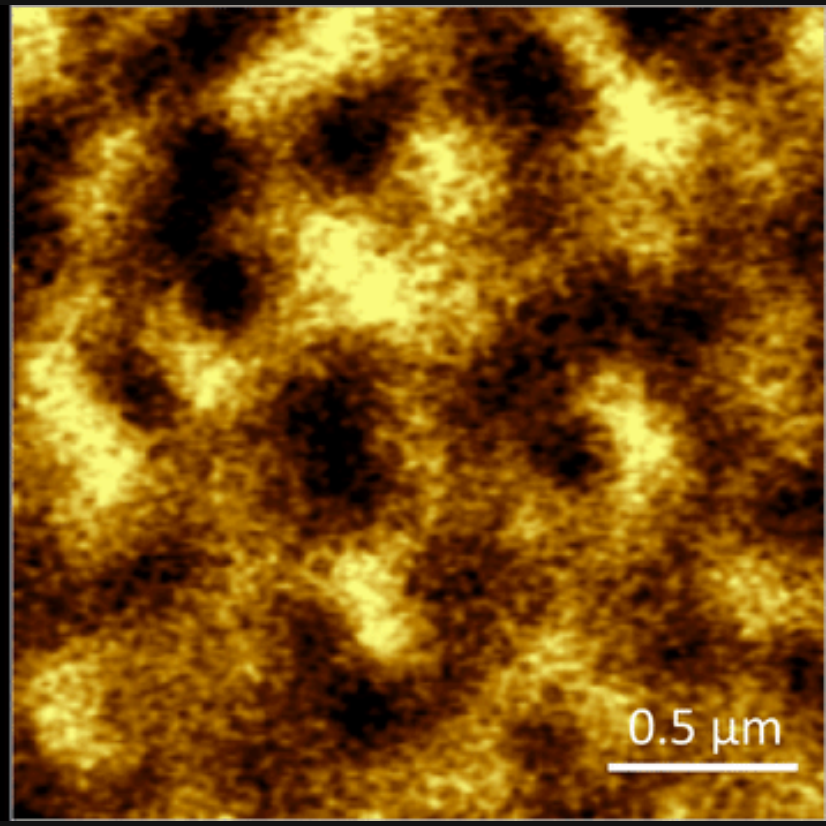
Presented by **SEAGATE.COM**

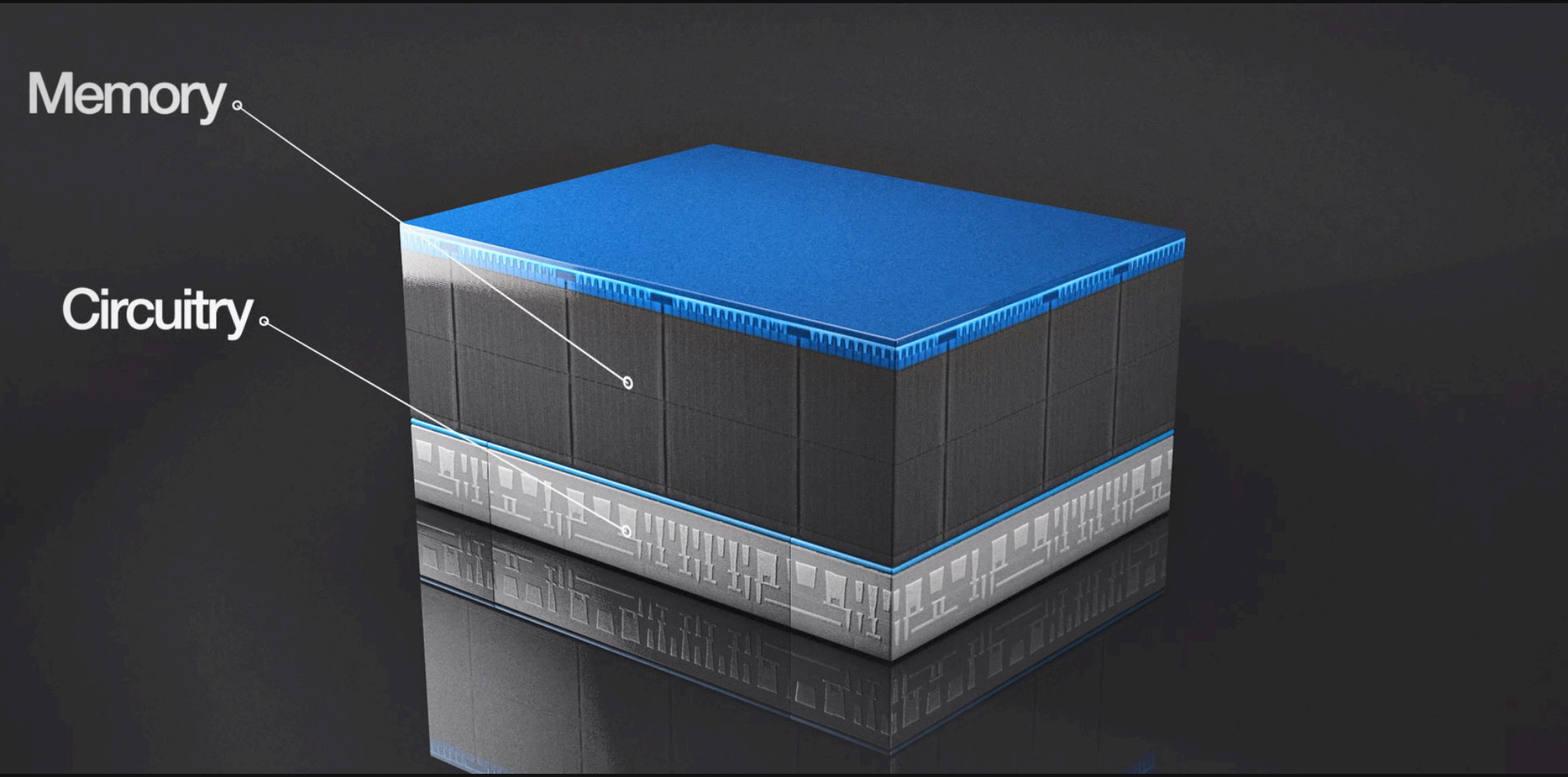
Designed & research by **ANIMAGRAFFS**

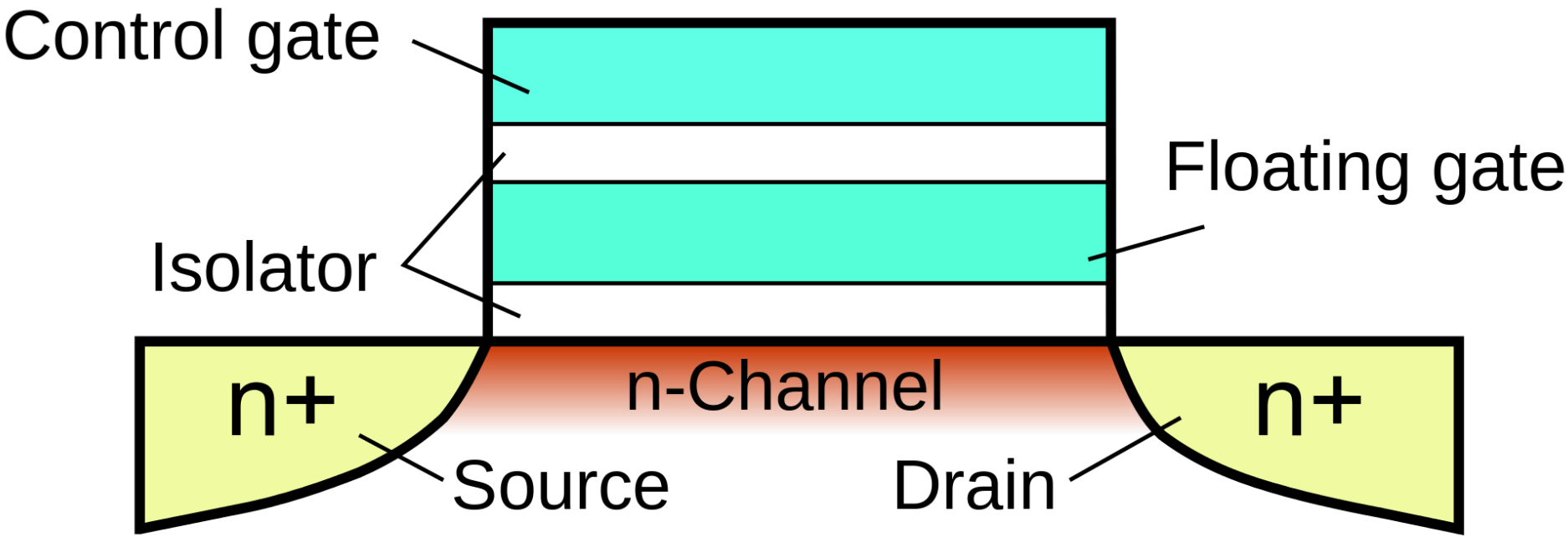
Hard Disk Drives (HDDs) use spinning disks (platters) to magnetically store information.



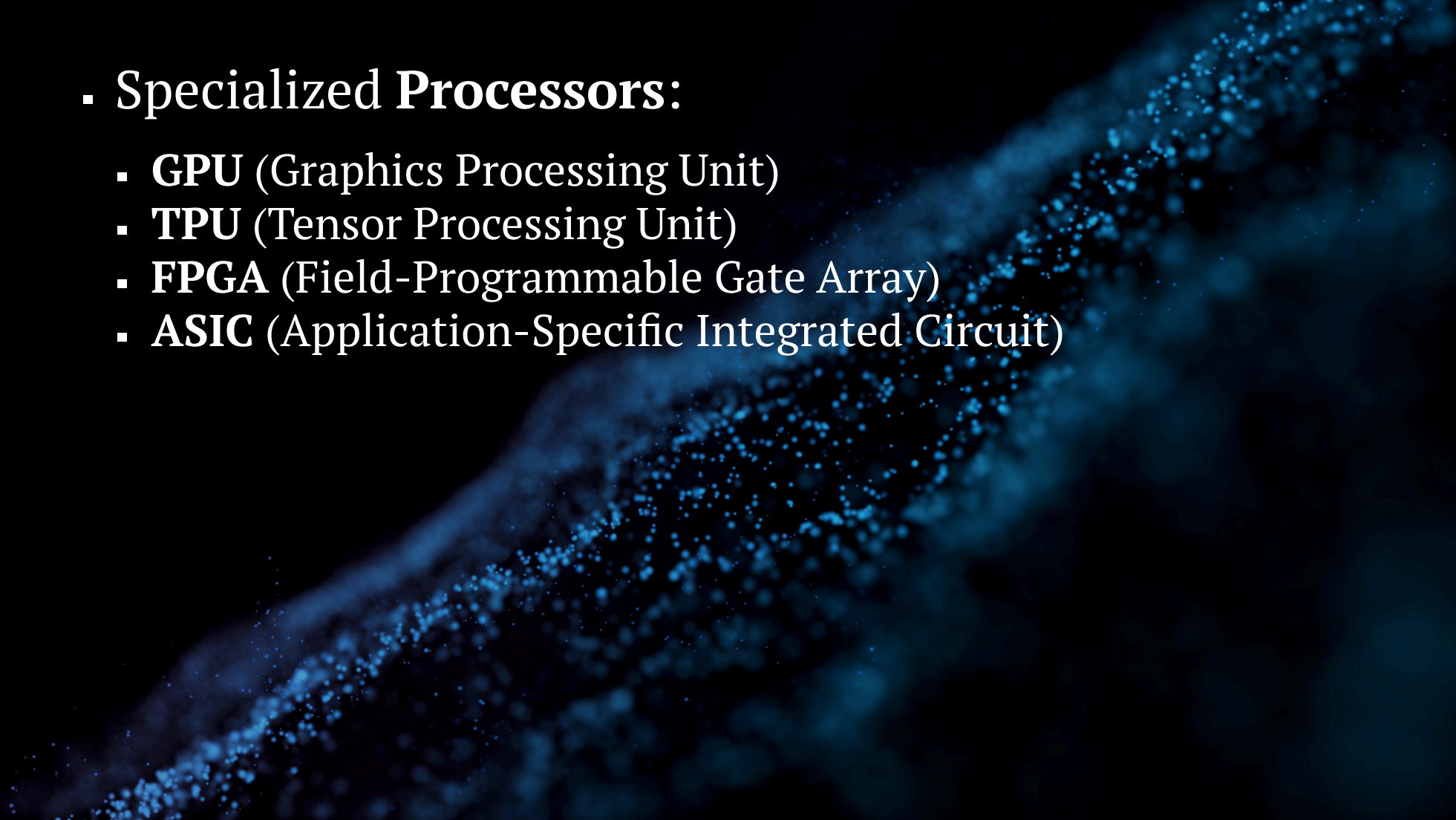

0.5 µm


0.5 µm


Memory
Circuitry

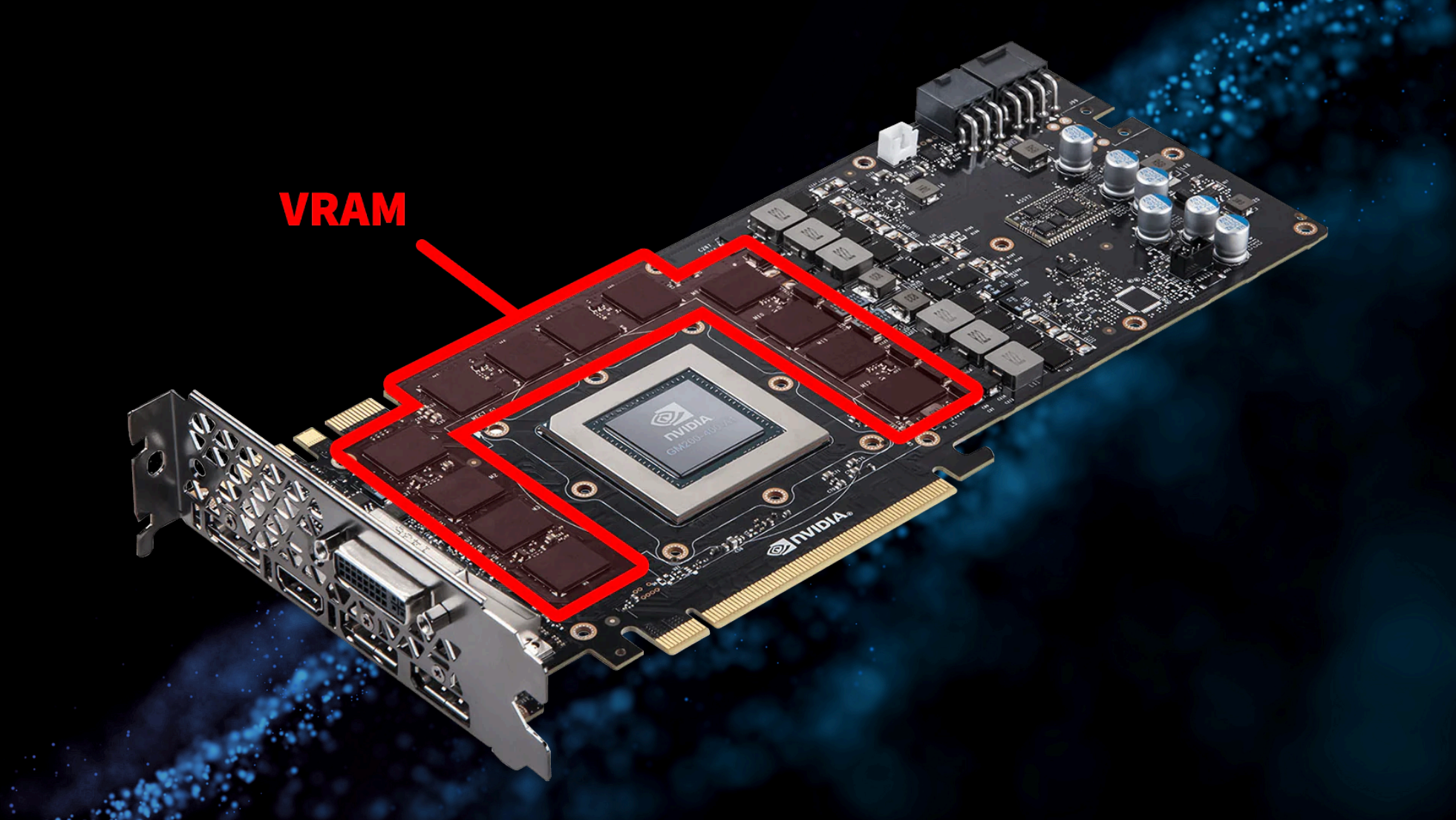

Control gate
Floating gate
Isolator
n+
n-Channel
n+
Source
Drain


Input/Output (I/O) Devices



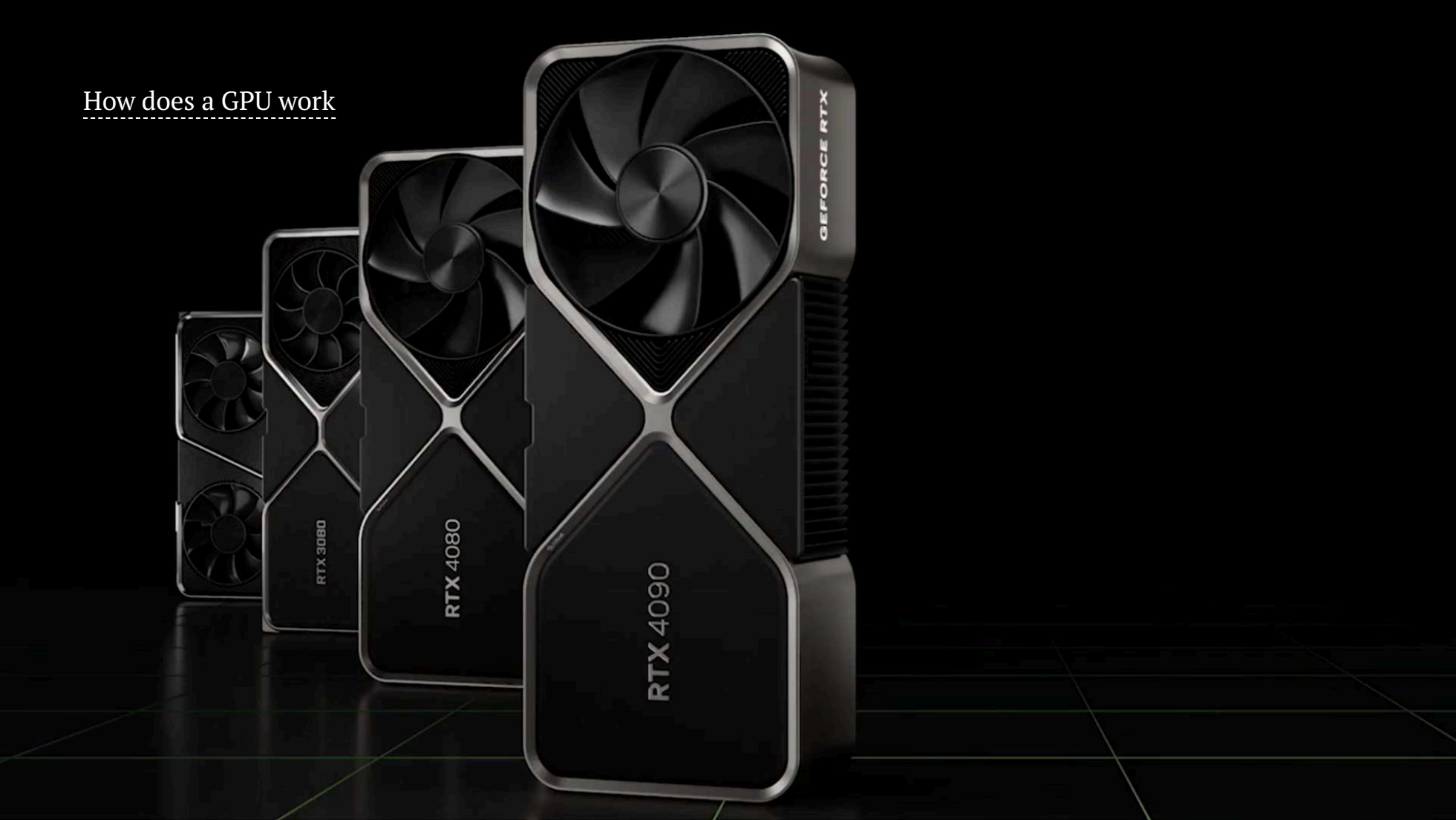
- Specialized **Processors**:
  - **GPU** (Graphics Processing Unit)
  - **TPU** (Tensor Processing Unit)
  - **FPGA** (Field-Programmable Gate Array)
  - **ASIC** (Application-Specific Integrated Circuit)

- **GPU** (Graphics Processing Unit)
  - Graphics Rendering
  - Parallel Processing Power
  - Accelerating Machine Learning and AI
  - Scientific and Data Analysis Computing
  - Video Processing and Encoding


VRAM
NVIDIA
NVIDIA

press

How does a GPU work




Power and Cooling
Networking
Monitoring and Management Tools


Security
Software
Virtualization and Cloud Computing

- **Software** Components
  - Operating Systems (**OS**)
    - Windows
    - macOS
    - Linux
  - **Middleware** and Virtualization
  - **Application** Software

Dr. Mindaugas Šarpis

Lessons on **Data Analysis** from **CERN**

Lecture 7:
Crash Course on Python Programming

Useful Resources for starting with Python

- Python Official Documentation
- Python Tutorial
- Python for Data Science Handbook
- Free introductory courses on codecademy, coursera, udemy, etc.

Why Python

- Python is one of the most popular and easy to learn programming languages in the world
- A large community of developers and users as well as a large number of libraries and frameworks make it a very versatile language
- Python itself and many of its libraries and tools are open-source and free to use and at the same time much more powerful than many commercial tools

Python Basics

- **Running python**
  - Interactive mode: `python` or `ipython`
  - Script mode: `python script.py`
  - Use `print("Hello, World!")` to output text
- **Comments**

```
# This line does ...
```

```
"""
This is a
multi-line
 comment
"""
```

Variables and Data Types

```
x          = 10       # Integer
y          = 3.14     # Float
name       = "Alice"  # String
is_valid = True       # Boolean
```

Basic Operators

```
# Arithmetic
2 + 3  # Addition
5 - 2  # Subtraction
3 * 4  # Multiplication
10 / 2  # Division
5 % 2  # Modulus
```

```
# Comparison
3 > 2  # True
4 == 4  # True
5 != 3  # True
```

Strings

```
s = "Hello, World!"
print(s[0])  # H (indexing)
print(s[-1])  # ! (negative indexing)
print(s[0:5])  # Hello (slicing)
print(len(s))  # Length of string
print(s.lower())  # Convert to lowercase
print(s.upper())  # Convert to uppercase
print(s.replace("World", "Python")) # Replace
```

Lists

```
numbers = [1, 2, 3, 4, 5]
print(numbers[0])  # First element
numbers.append(6)  # Add element
numbers.remove(3)  # Remove element
print(len(numbers))  # Length of list
numbers.sort()  # Sort list
```

- Lists are mutable and can hold mixed data types

```
customer_info = ["Alice", 25, "New York", "Premium", True]
```

Dictionaries

```
person = {"name": "Alice", "age": 25}
print(person["name"])  # Alice
person["age"] = 26  # Update value
person["city"] = "New York"  # Add key-value pair
print(person.keys())  # Get all keys
print(person.values())  # Get all values
```

- Dictionaries store key-value pairs and are mutable

Control Flow: Conditional Statements

```
x = 10
if x > 5:
    print("x is greater than 5")
elif x == 5:
    print("x is 5")
else:
    print("x is less than 5")
```

Control Flow: Loops

```
# For loop
for i in range(5):
    print(i)  # Prints 0 to 4

# While loop
x = 0
while x < 5:
    print(x)
    x += 1
```

Functions

```
def greet(name):
    return f"Hello, {name}!"
print(greet("Alice"))
```

- Functions help organize code into reusable blocks
- Use `return` to return a value from a function -->

Exception Handling

```
try:
    x = 10 / 0
except ZeroDivisionError:
    print("Cannot divide by zero!")
finally:
    print("Execution completed")
```

- Use `try` and `except` to handle errors
- `finally` block always executes

Modules and Imports

```python
import math
print(math.sqrt(16))  # 4.0

from random import randint
print(randint(1, 10))  # Random number between 1 and 10
```

- Use `import` to bring in external modules
- `from module import function` imports specific functions

File Handling

```python
# Writing to a file
with open("test.txt", "w") as file:
    file.write("Hello, Python!")

# Reading from a file
with open("test.txt", "r") as file:
    content = file.read()
    print(content)
```

- Use `with open()` to handle file operations safely
- `"r"` for reading, `"w"` for writing, `"a"` for appending

Dr. Mindaugas Šarpis

Lessons on **Data Analysis** from **CERN**

Lecture 8:
Markdown

Markdown

- **Lightweight markup language** with plain text formatting syntax
- **Converts** plain text to **HTML**
- **Easy to read** and **write**
- **Simple** and **intuitive**

Markdown Syntax

Headers

```
# Header 1
## Header 2
### Header 3
#### Header 4
##### Header 5
###### Header 6
```

- Use `#` for different header levels:

Header 1

Header 2

Header 3

Header 4

Header 5

HEADER 6

Emphasis

Italic, Bold, and Strikethrough

```
*Italic* or _Italic_
**Bold** or __Bold__
~~Strikethrough~~
```

Italic

Bold

~~Strikethrough~~

Lists

Unordered List

```
- Item 1
- Item 2
  - Subitem 1
  - Subitem 2
```

- Item 1
- Item 2
  - Subitem 1
  - Subitem 2

Ordered List

```
1. First
2. Second
3. Third
```

1. First
2. Second
3. Third

Task Lists

```
- [x] Task 1
- [ ] Task 2
- [ ] Task 3
```

- [x] Task 1
- [ ] Task 2
- [ ] Task 3

Links

```
[OpenAI](https://openai.com)
```

Images

```
![Markdown Logo](https://upload.wikimedia.org/wi
```

OpenAI

Code Blocks

Inline Code

```
Use `inline code` within a sentence.
```

Code Block

- Use triple backticks to create "```" a code block

```
  function hello() {
  console.log("Hello, world!");
  }
```

Syntax Highlighting

- Add the language name after the first set of backticks
- For example, `python`

```python
def hello():
    print("Hello, world!")
```

Blockquotes

```
> This is a blockquote.
> It can span multiple lines.
```

> This is a blockquote. It can span multiple lines

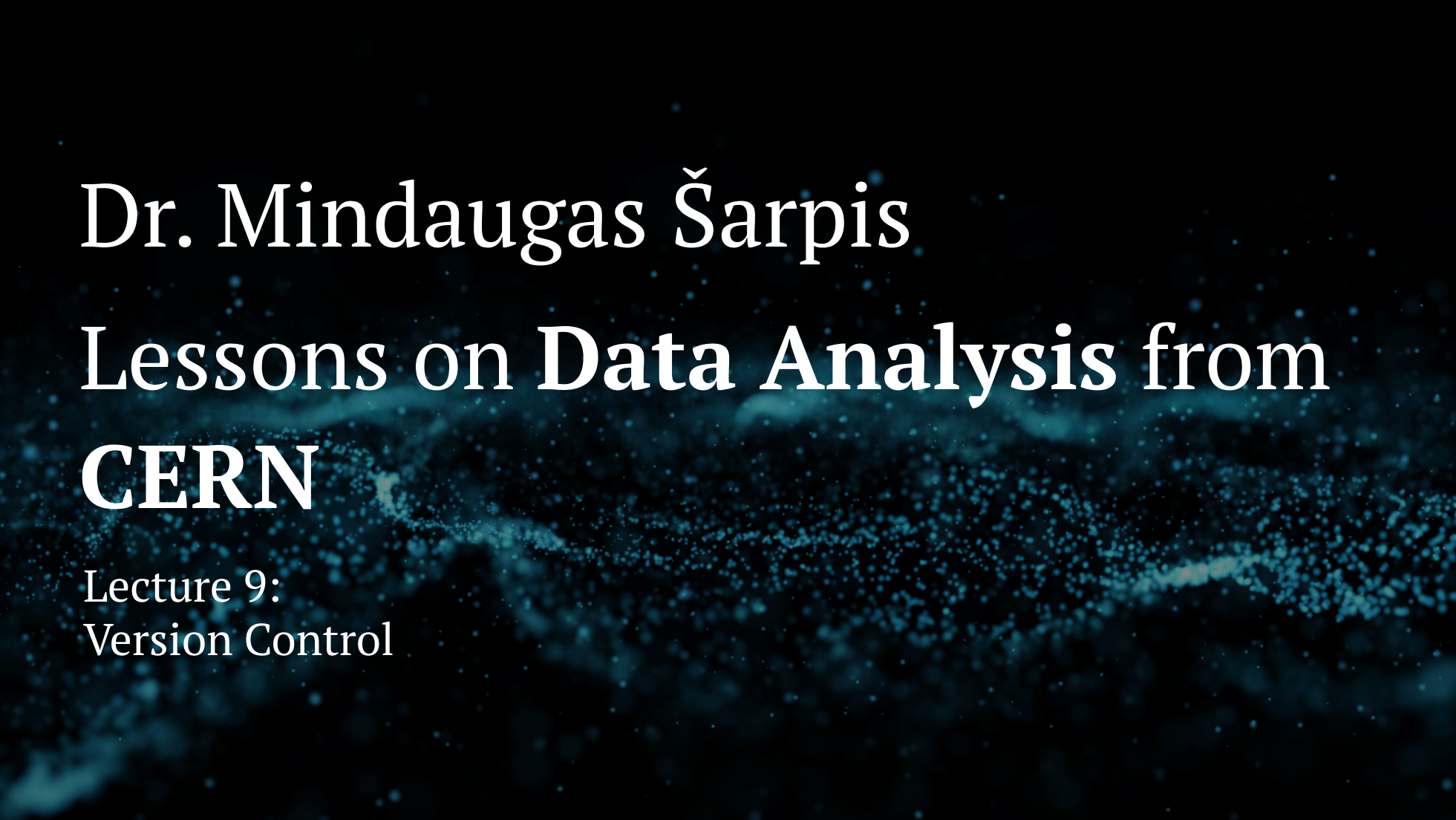
Horizontal Rule

```
---
```

Tables

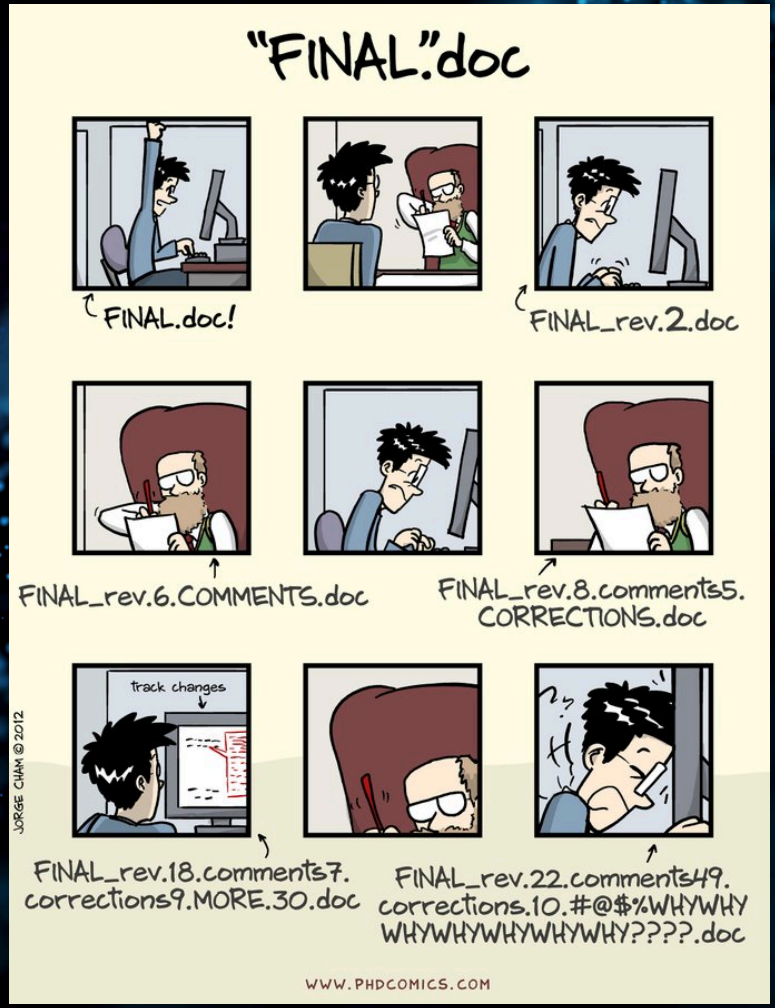
```
| Syntax | Description |
|--------|-------------|
| Header | Title       |
| Cell   | Data        |
```

Syntax	Description
Header	Title
Cell	Data


Dr. Mindaugas Šarpis
Lessons on **Data Analysis** from **CERN**
Lecture 9:
Version Control

The Importance of Version Control

- Even if working alone, many different version of the same file will exist
- Some overwritten changes might be needed later
- A "versioned" file might be needed when implementing comments from supervisor / reviewers
- This hold true for written work, code and other files



Tracking Changes (differences)

- Rather than saving multiple copies of the same file, we can track changes
- Word processors and other software have some change-tracking functionality but it is limited (no synchronous editing, no change history, etc.)
- `git` is an open-source version control system that is used to track changes in files

Different Versions

- An eddit to a file might overwrite some of the content in the previous version
- This *divergences* may arrise while working alone, but they are really common when multiple people are working on the same file

Merging

- `git` has great functionality for merging different versions of the same file
- If the previous content is not overwritten, or deleted, merge just combines the changes into one file
- If changes over-write each other a so-called **merge conflict** arises

Using `git` for the first time

- The user name and email address need to configured.

```
git config --global user.name "Mindaugas Sarpis"
git config --global user.email "mindaugas.sarpis
```

- Check the configuration with:

```
git config --list
```

- Edit the configuration with:

```
git config --global --edit
```

- Open configuration help:

```
git config --h
git config --help
```

```
usage: git config [<options>]

Config file location
    --global              use global config file
    --system              use system config file
    --local               use repository config file
    --worktree            use per-worktree config file
    -f, --file < file >     use given config file
    --blob < blob-id >      read config from given blob obj

Action
    --get                 get value: name [value-pattern]
    --get-all             get all values: key [value-patter
    --get-regexp          get values for regexp: name-regex
    --get-urlmatch        get value specific for the URL: s
    --replace-all         replace all matching variables: n
    --add                 add a new variable: name value
    --unset               remove a variable: name [value-pa
    --unset-all           remove all matches: name [value-p
    --rename-section      rename section: old-name new-name
    --remove-section      remove a section: name
    -l, --list            list all
    --fixed-value         use string equality when comparin
    -e, --edit            open an editor
    --get-color           find the color configured: slot [
    --get-colorbool       find the color setting: slot [std
```

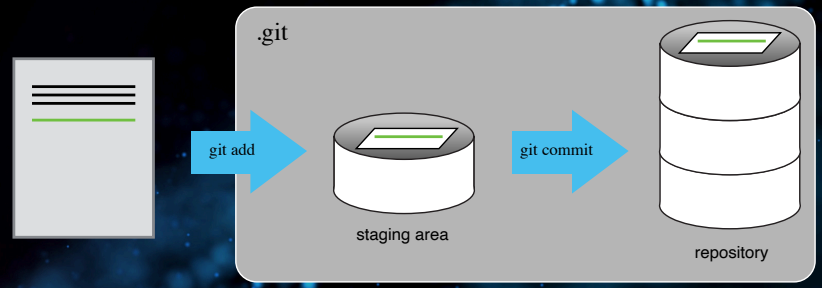
Creating a new repository

- A repository is initialized with the following command:

```
git init
```

- This command creates a new repository in the current directory.
- The repository is a hidden directory called `.git` that contains all the information changes tracked by `git`.
- You can check the status of the repository with:

```
git status
```



- The repository is empty at this point and the output will be:

```
On branch main

No commits yet

nothing to commit (create/copy files and use "git add" to t
```

Staging Area

- `git` has a staging area where files are placed to track the changes made to them.
- To move a file to the staging area use:

```
git add <file>
```

- To move all files to the staging area use:

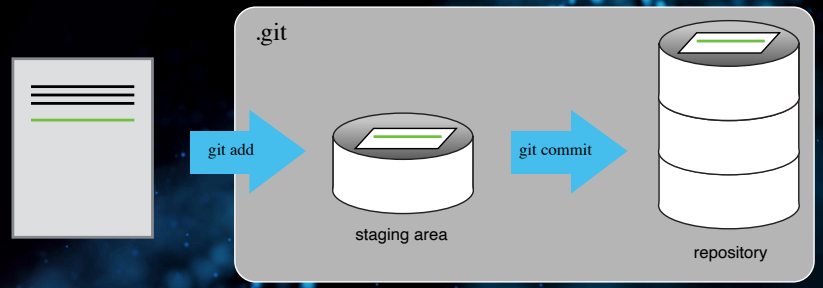
```
git add --all
```

- To unstage a file use:

```
git restore --staged <file>
```

- Changes to files can be viewed with:

```
git diff
```



- When staged files are present, the output of `git status` will be:

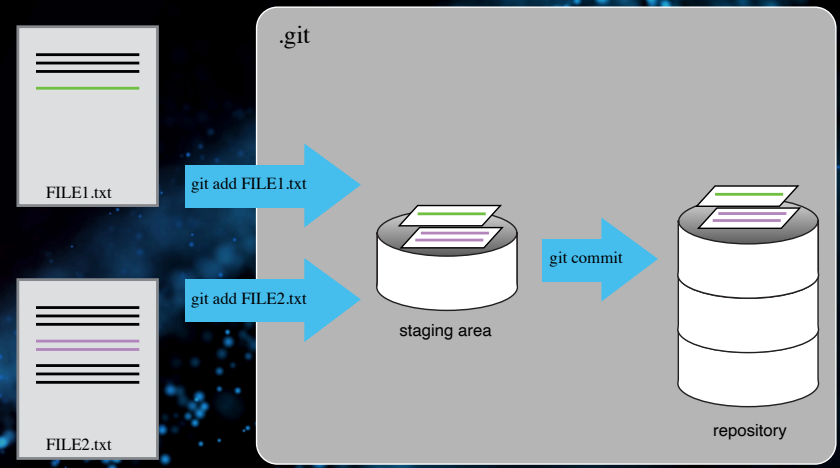
```
On branch main
Your branch is up to date with 'origin/main'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   < file >
```

Committing Changes

- Files are committed to the repository from the staging area with:

```
git commit -m "A message describing the changes"
```

- Commit is a snapshot of the repository at a given time
- Only changes to files are tracked, not the directories themselves
- It's best to keep the commits small and focused on a single change
- The commit message should be descriptive and concise
- The commit message should be in the present tense



Restoring Changes

- Changes to files can be restored to the last commit with:

```
git restore < file >
```

- Changes to files can be restored to the last commit and the staging area with:

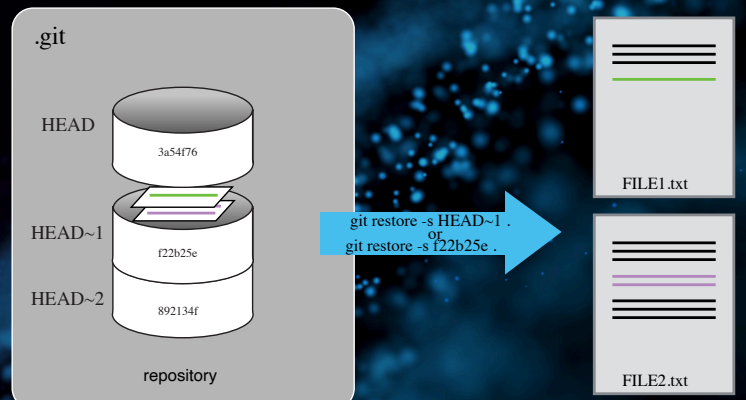
```
git restore --staged < file >
```

- Changes to files from previous commits can be restored using the *hash* of the commit:

```
git restore --source=<hash> < file >
```

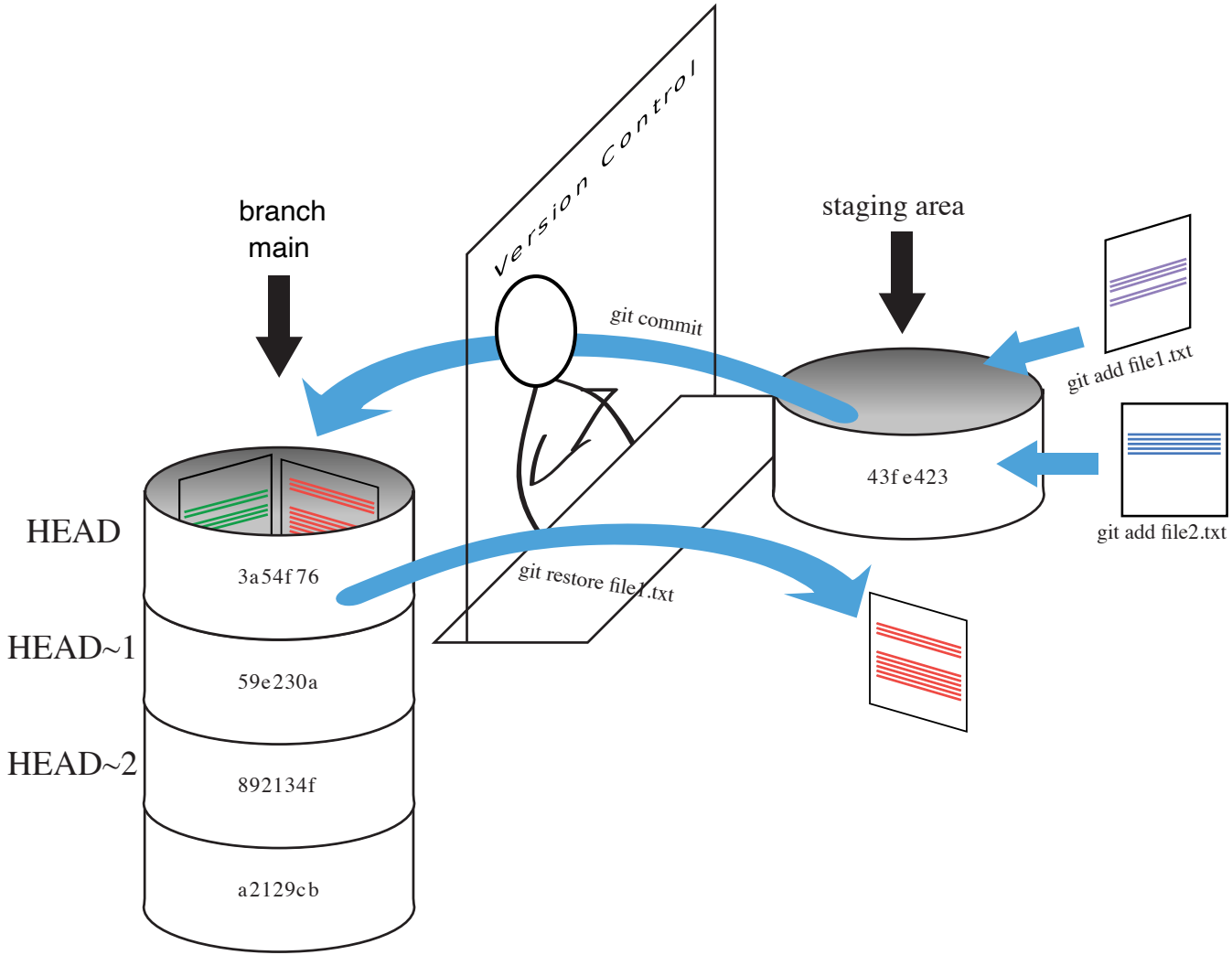
- A new commit reverting the changes can be made with:

```
git revert < hash >
```



- The entire repository can be restored to the last commit with deleting the changes:

```
git reset --hard < hash >
```


branch
main
Version Control
staging area
git commit
git add file1.txt
git add file2.txt
43f e423
HEAD
3a54f 76
HEAD~1
59e230a
HEAD~2
892134f
a2129cb
git restore file1.txt

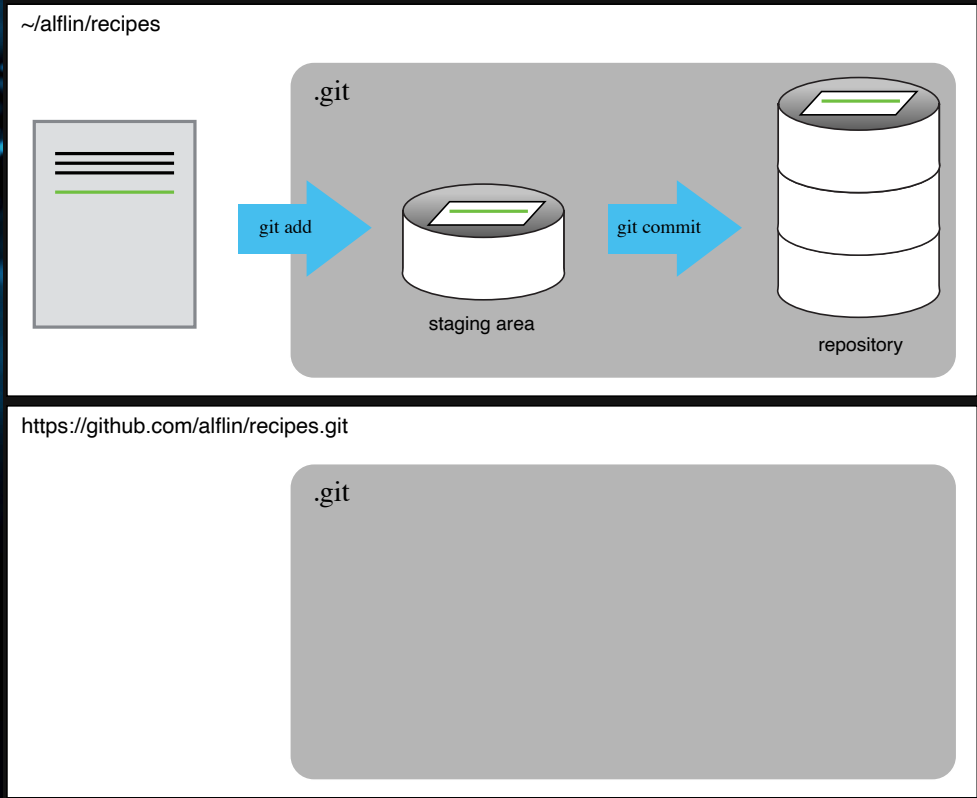
Ignoring Files and Directories

- There might be files that you don't want to track with `git`
  - Temporary files
  - Output files
  - Files with sensitive information
  - Large files
- These files can be ignored by creating a `.gitignore` file in the repository

```
# Byte-compiled / optimized / DLL files
__pycache__/
*.py[cod]
*$py.class

# C extensions
*.so

# Distribution / packaging
.Python
build/
develop-eggs/
dist/
downloads/
eggs/
.eggs/
lib/
lib64/
parts/
```

Git Remotes

- One of the most powerful features of `git` is the ability to work with remote repositories.
- Remote repositories are copies of the repository that are stored on a server.
- Using one of the remote providers (GitHub, GitLab, Bitbucket, etc.) you can store your repository in the cloud.
- This enables collaboration with other people and provides a backup of your work.



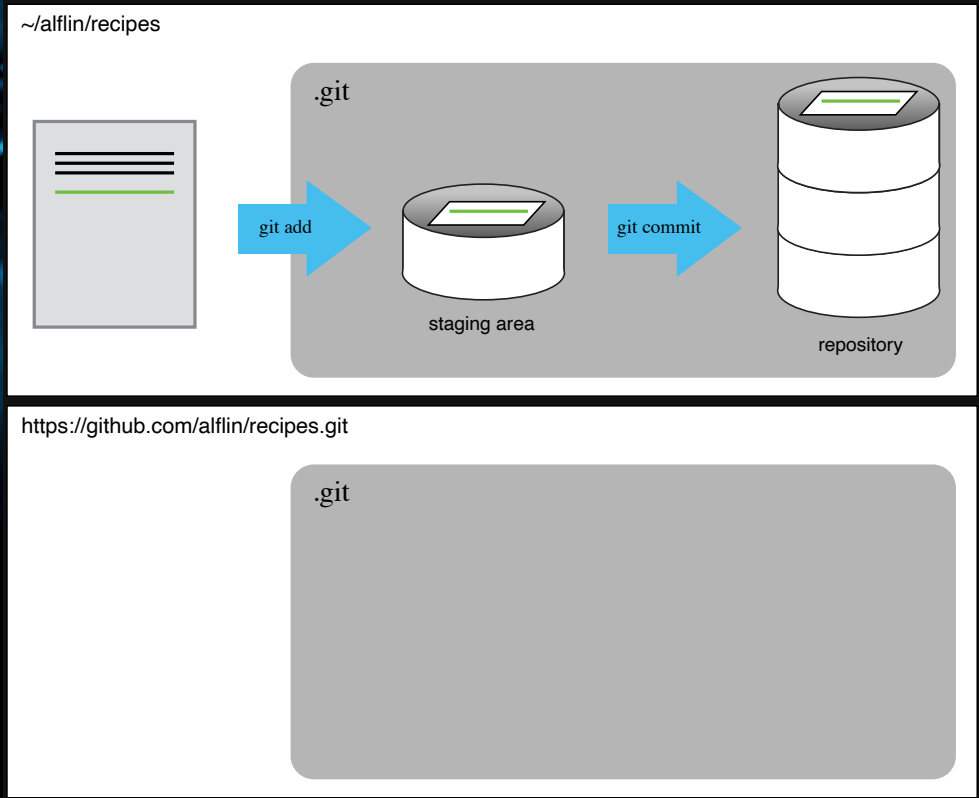
Git Remotes

- The remote is created via the remote provider (GitHub, GitLab, Bitbucket, etc.).
- A remote URL needs to be added to the local repository with:

```
git remote add origin git@github.com:mygithub/myremo
```

- To check which remotes are added:

```
git remote -v
```



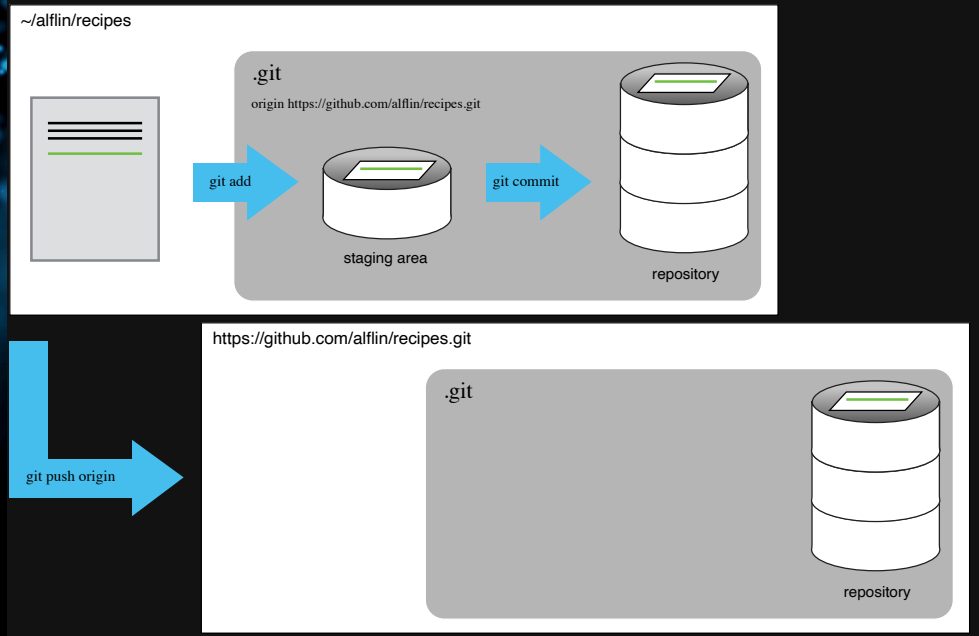
Push / Pull Operations

- Changes to the local repository can be pushed to the remote repository with:

```
git push origin main
```

- Changes to the remote repository can be pulled to the local repository with:

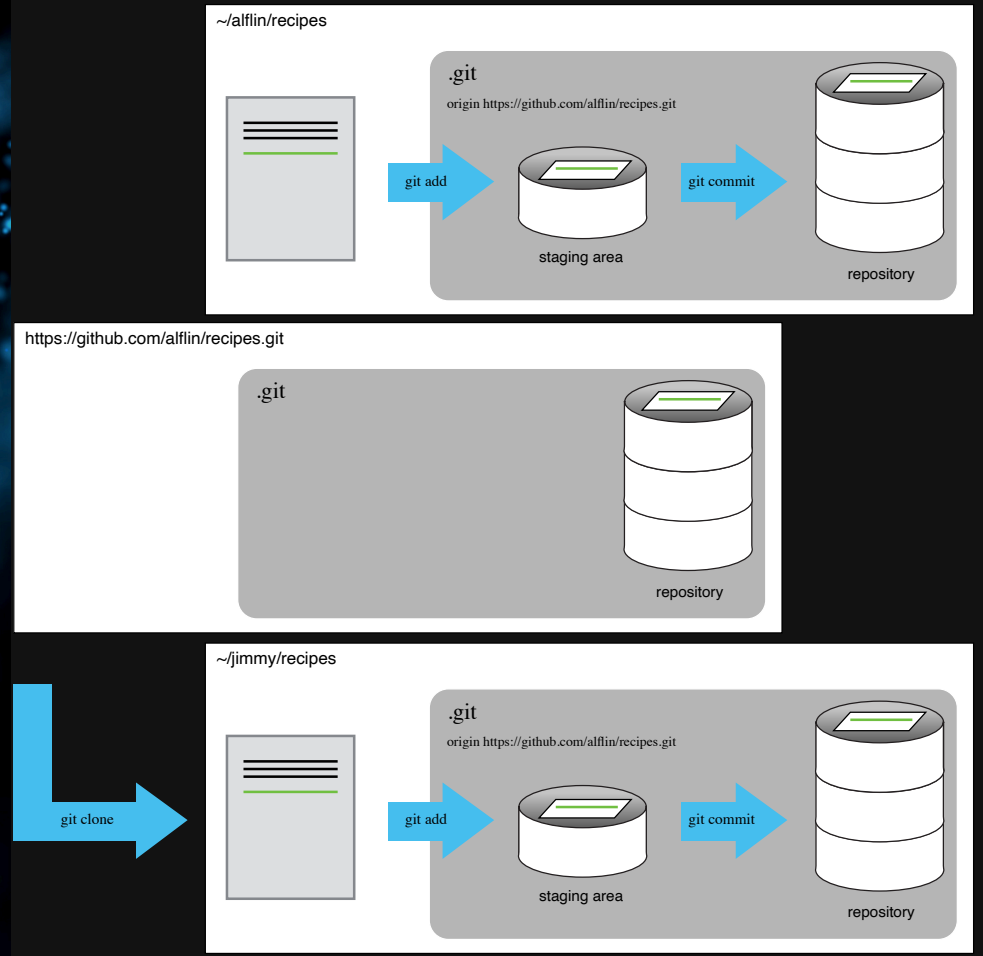
```
git pull
```



Cloning Repositories

- A repository can be cloned from a remote repository with:

```
git clone < URL >
```



Branches

- `git` has a powerful branching system that allows for multiple versions of the repository to be worked on simultaneously.
- The default branch is called `main`.
- A new branch can be created with:

```
git branch < branch-name >
```

- The branch can be switched with:

```
git checkout < branch-name >
```

Dr. Mindaugas Šarpis

Lessons on **Data Analysis** from **CERN**

Lecture 10:
Data Visualization

Inspired by: C. O. Wilke Fundamentals of Data Visualization

Data **Visualization**

- One of the key components of data analysis
  - Interpreting Results
  - Reporting Findings
- Chosing appropriate method of visualization and representation is crucial

Arc Diagram

Area Graph

Bar Chart

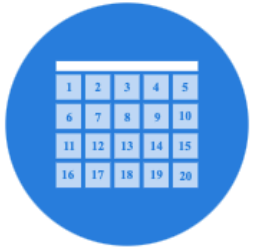
Box & Whisker Plot

Brainstorm

Bubble Chart

Bubble Map

Bullet Graph



Calendar

Candlestick Chart

Chord Diagram

Choropleth Map

Circle Packing

Connection Map

Density Plot

Donut Chart

Dot Map

Dot Matrix Chart

Circle Packing

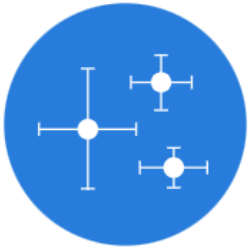
Connection Map

Density Plot

Donut Chart

Dot Map

Dot Matrix Chart



Error Bars

Flow Chart

Flow Map

Gantt Chart

Heatmap

Histogram

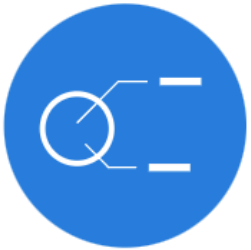


Illustration Diagram

Kagi Chart

Line Graph

Marimekko Chart

Multi-set Bar Chart

Network Diagram

Pie Chart

Point & Figure Chart

Population Pyramid

Proportional Area Chart

Radar Chart

Radial Bar Chart

Radial Column Chart

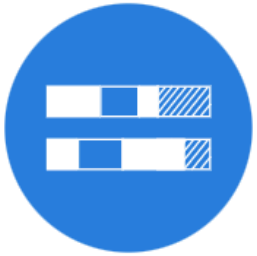
Sankey Diagram

Scatterplot

Span Chart

Spiral Plot

Stacked Area Graph



Stacked Bar Graph

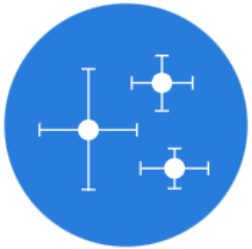
Stem & Leaf Plot

Stream Graph

Sunburst Diagram

Tally Chart

Timeline



Error Bars

Flow Chart

Flow Map

Gantt Chart

Heatmap

Histogram

Illustration Diagram

Kagi Chart

Line Graph

Marimekko Chart

Multi-set Bar Chart

Network Diagram

Nightingale Rose Chart

Non-ribbon Chord Diagram

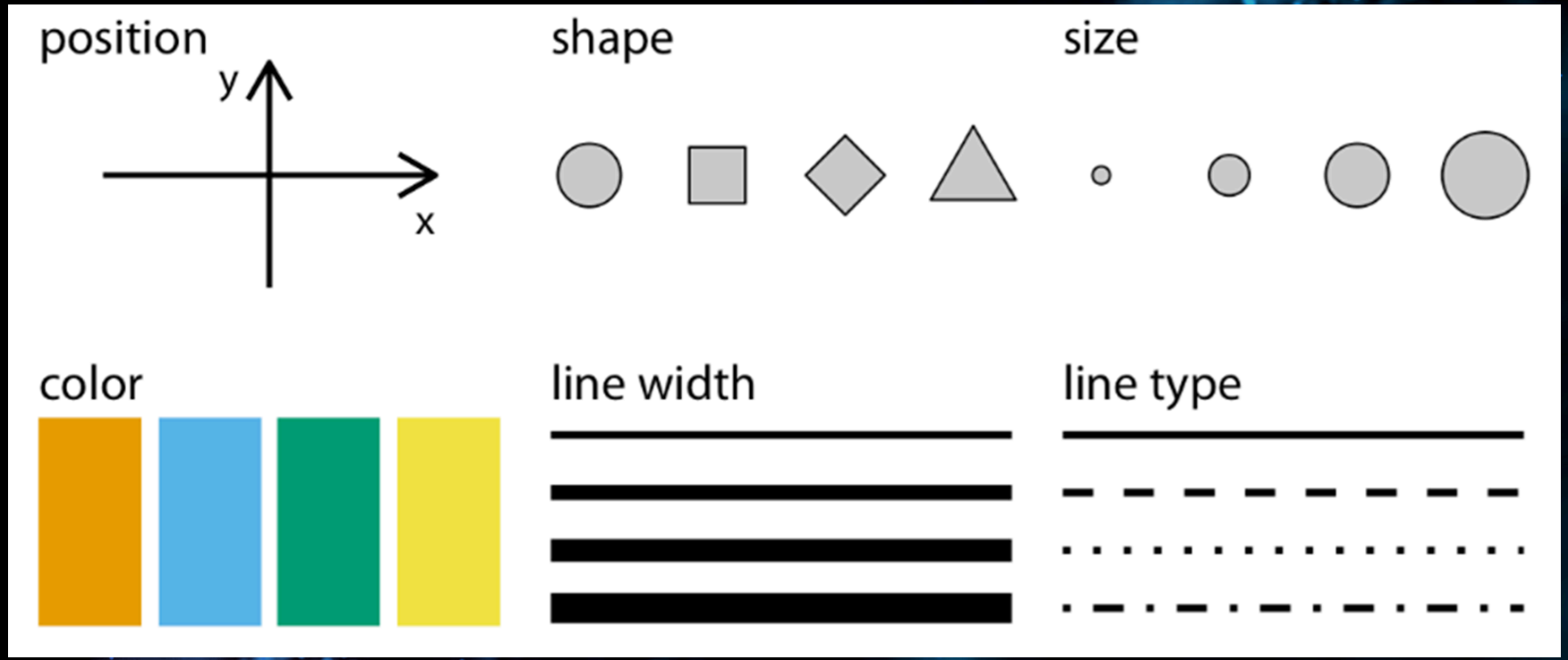
Open-high-low-close Chart

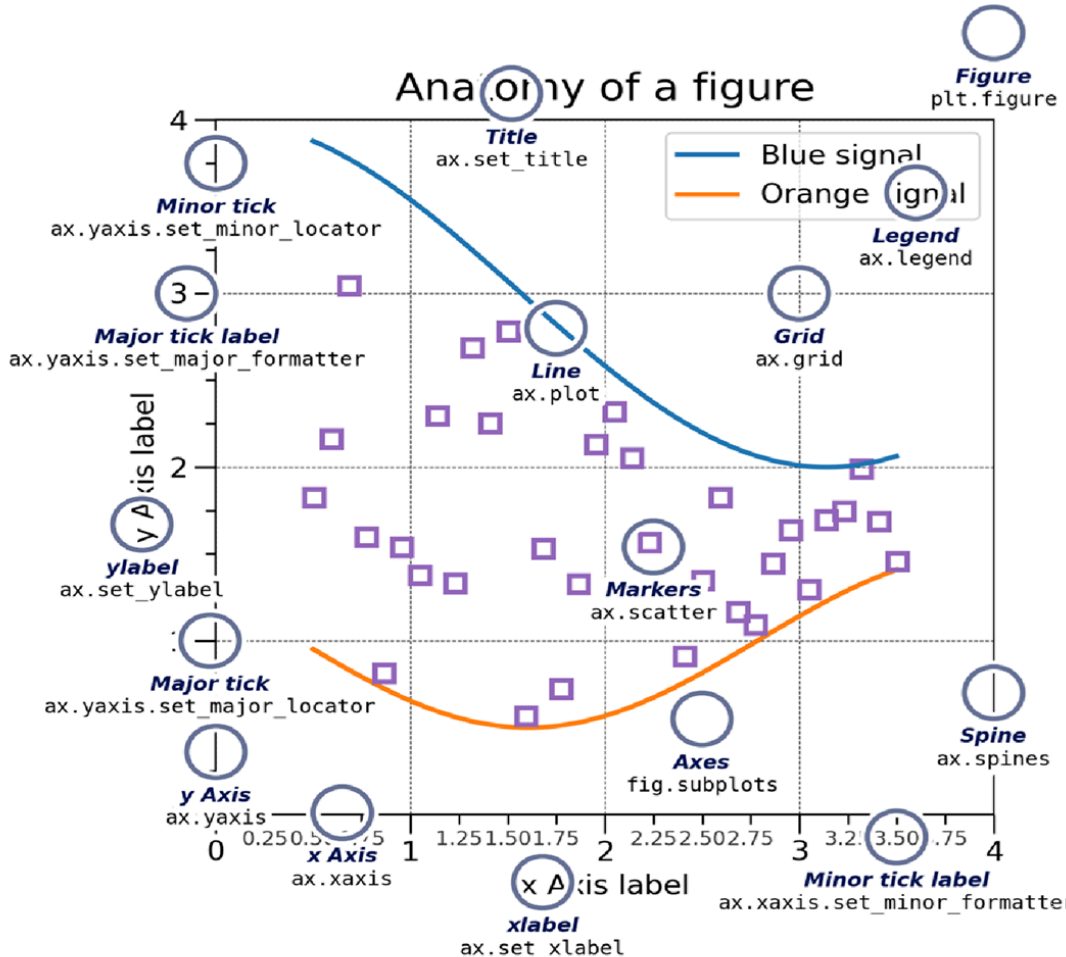
Parallel Coordinates Plot

Parallel Sets

Pictogram Chart

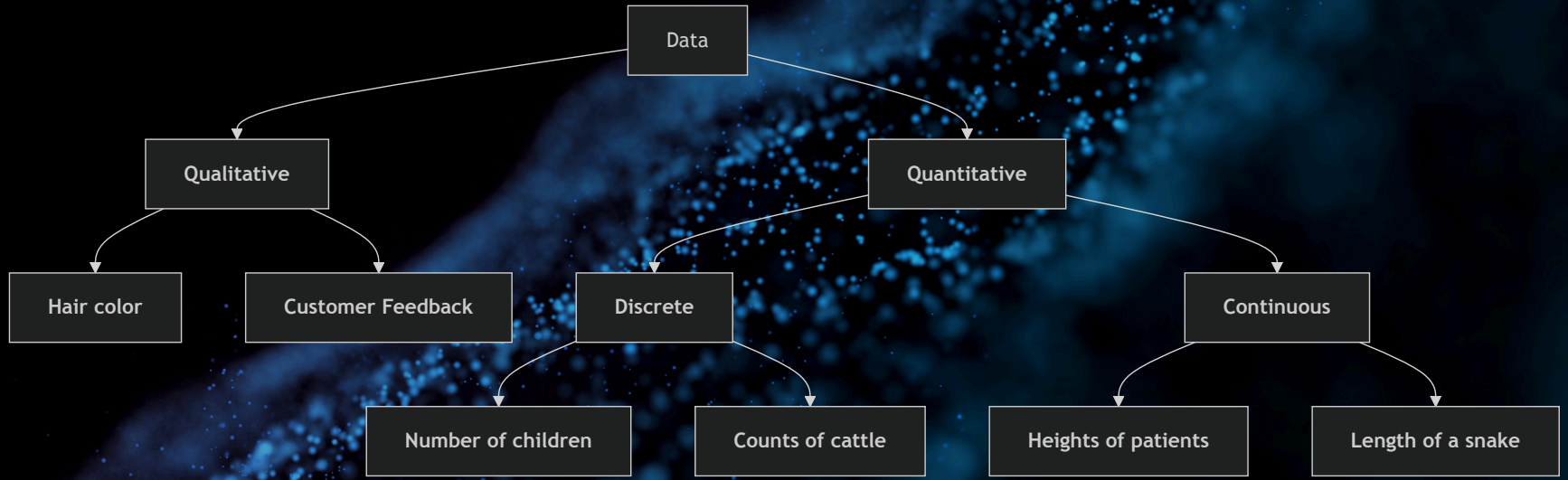
Aesthetics of Data Visualization




Anatomy of a figure
Figure
plt.figure
Title
ax.set_title
Blue signal
Orange signal
Legend
ax.legend
Minor tick
ax.yaxis.set_minor_locator
Major tick label
ax.yaxis.set_major_formatter
Grid
ax.grid
Line
ax.plot
y Axis label
ylabel
ax.set_ylabel
Markers
ax.scatter
Major tick
ax.yaxis.set_major_locator
Spine
ax.spines
Axes
fig.subplots
y Axis
ax.yaxis
x Axis
ax.xaxis
x Axis label
xlabel
ax.set_xlabel
Minor tick label
ax.xaxis.set_minor_formatter
0
1
2
3
4

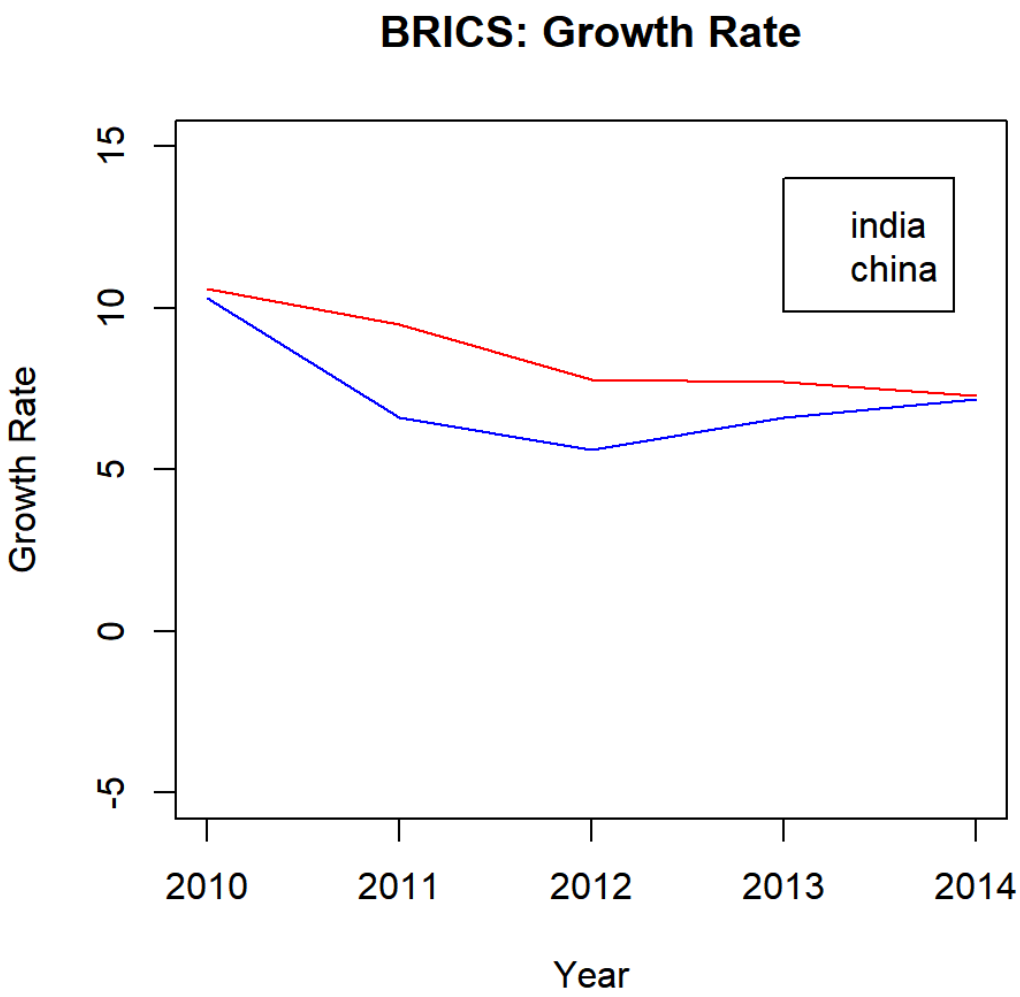
Continuous and **Discrete** Data

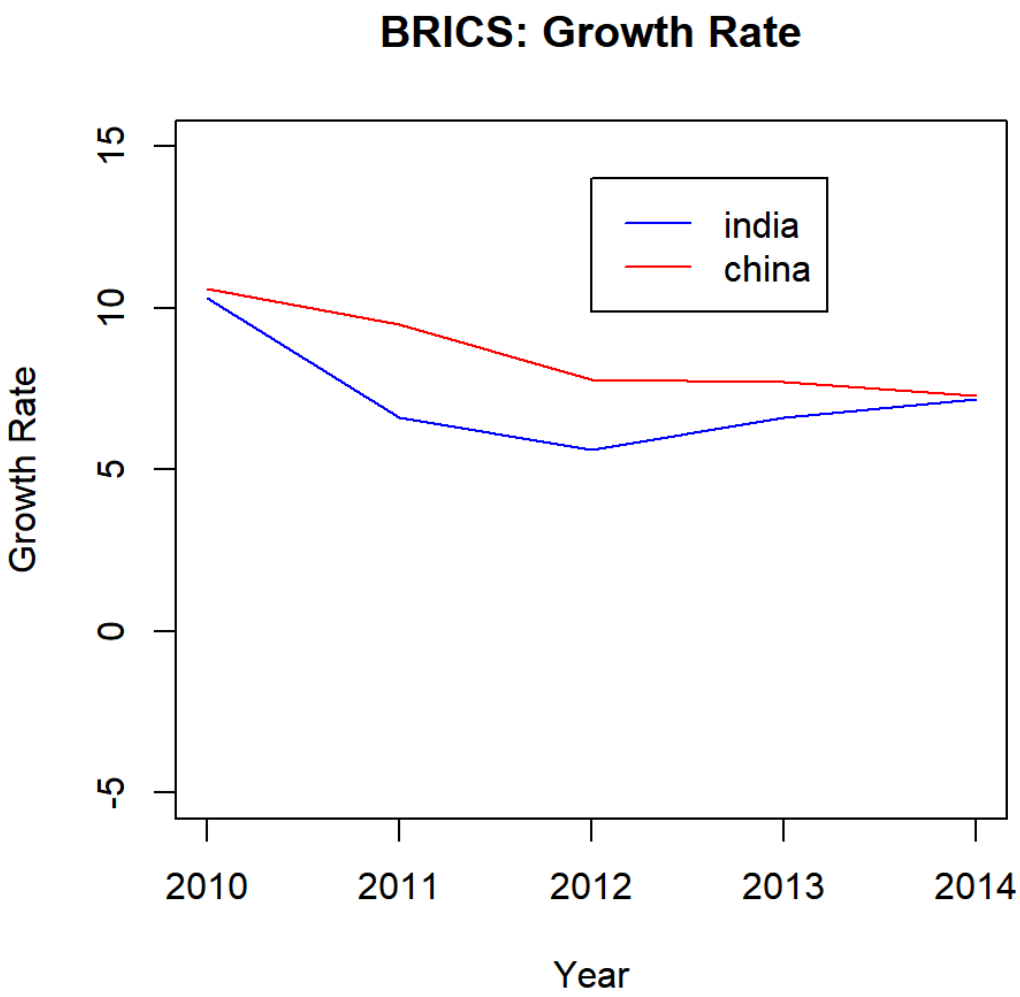
- Quantitative / Numerical Data
- Qualitative / Categorical Data
- Date and Time

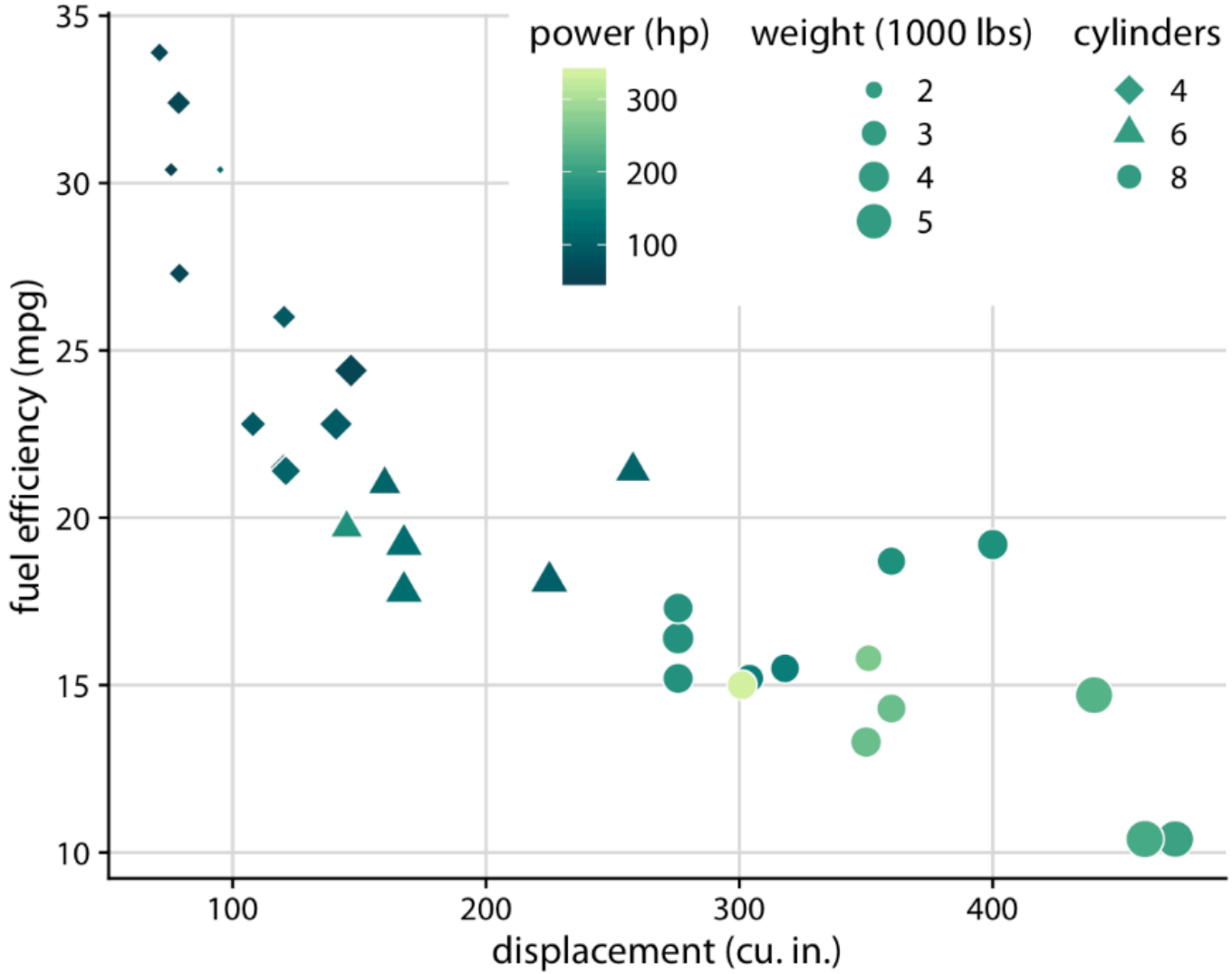


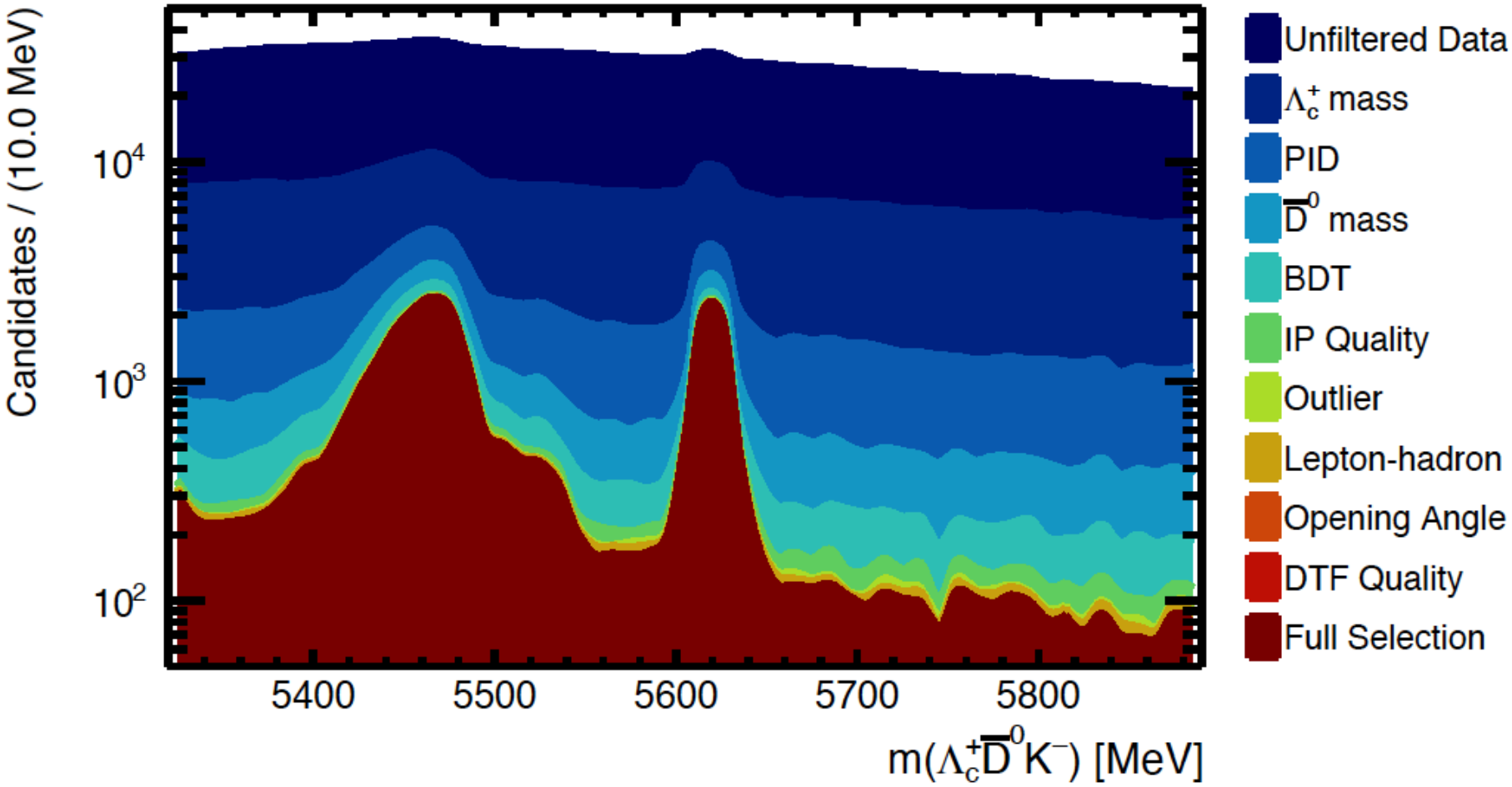
Legend

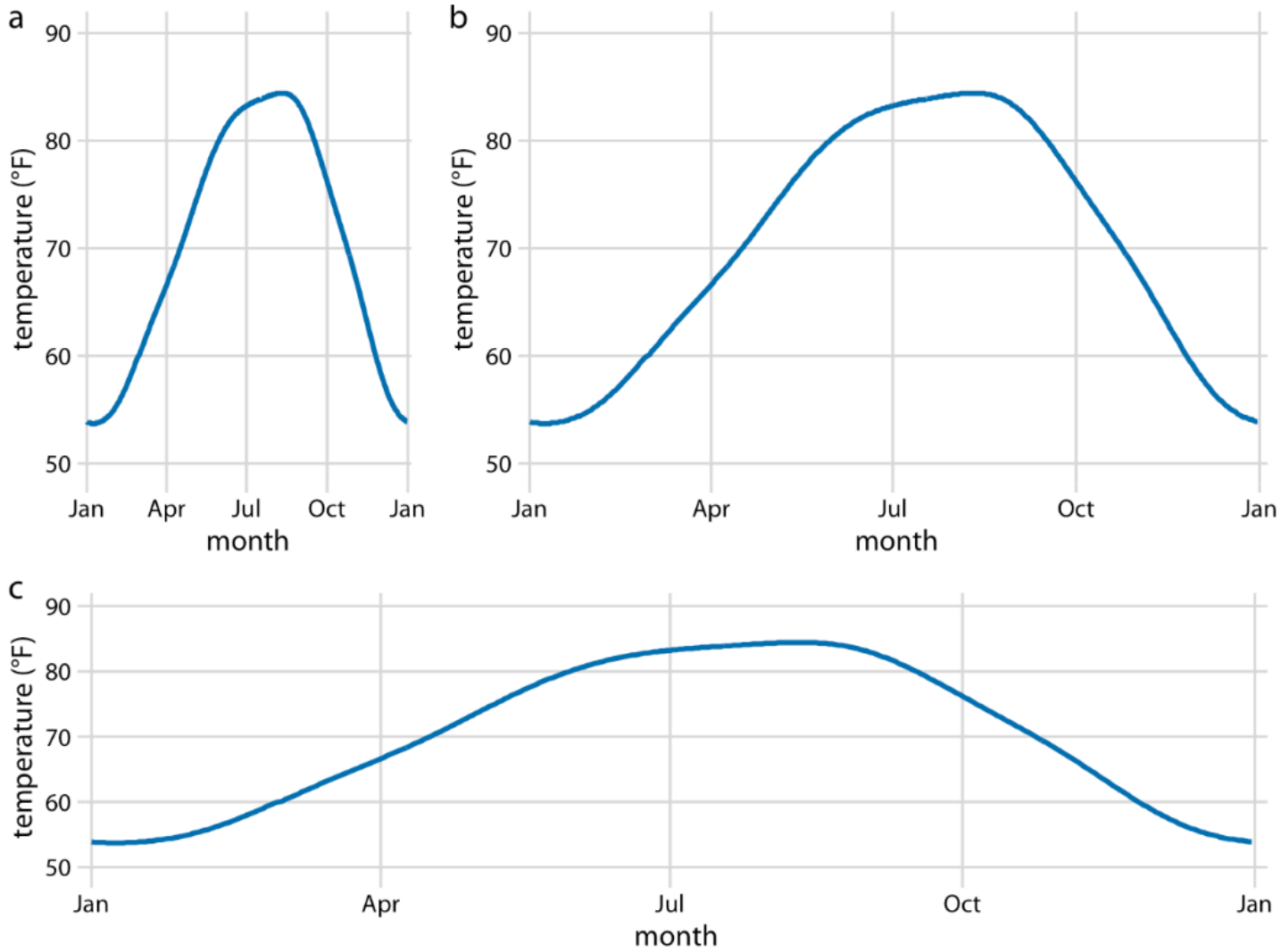
- Legend is a key component of a plot that explains the meaning of the data
- Legend might not be necessary if the data is self-explanatory (e.g. bar chart, line chart with one line, plot annotations)

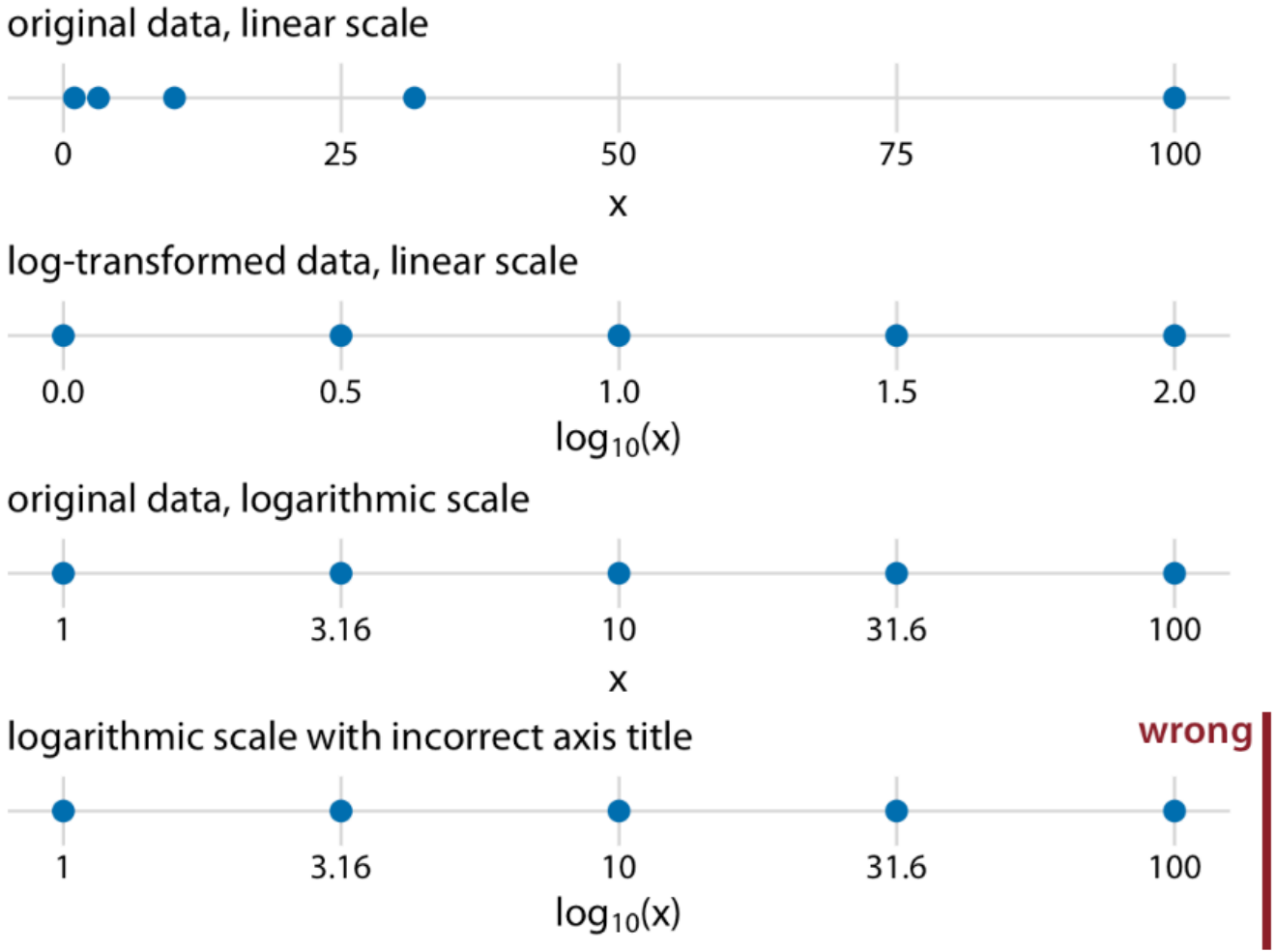

BRICS: Growth Rate
Growth Rate
15
10
5
0
-5
2010
2011
2012
2013
2014
Year
india
china


BRICS: Growth Rate
Growth Rate
15
10
5
0
-5
2010
2011
2012
2013
2014
Year
india
china


power (hp)
300
200
100
weight (1000 lbs)
2
3
4
5
cylinders
4
6
8
fuel efficiency (mpg)
35
30
25
20
15
10
100
200
300
400
displacement (cu. in.)

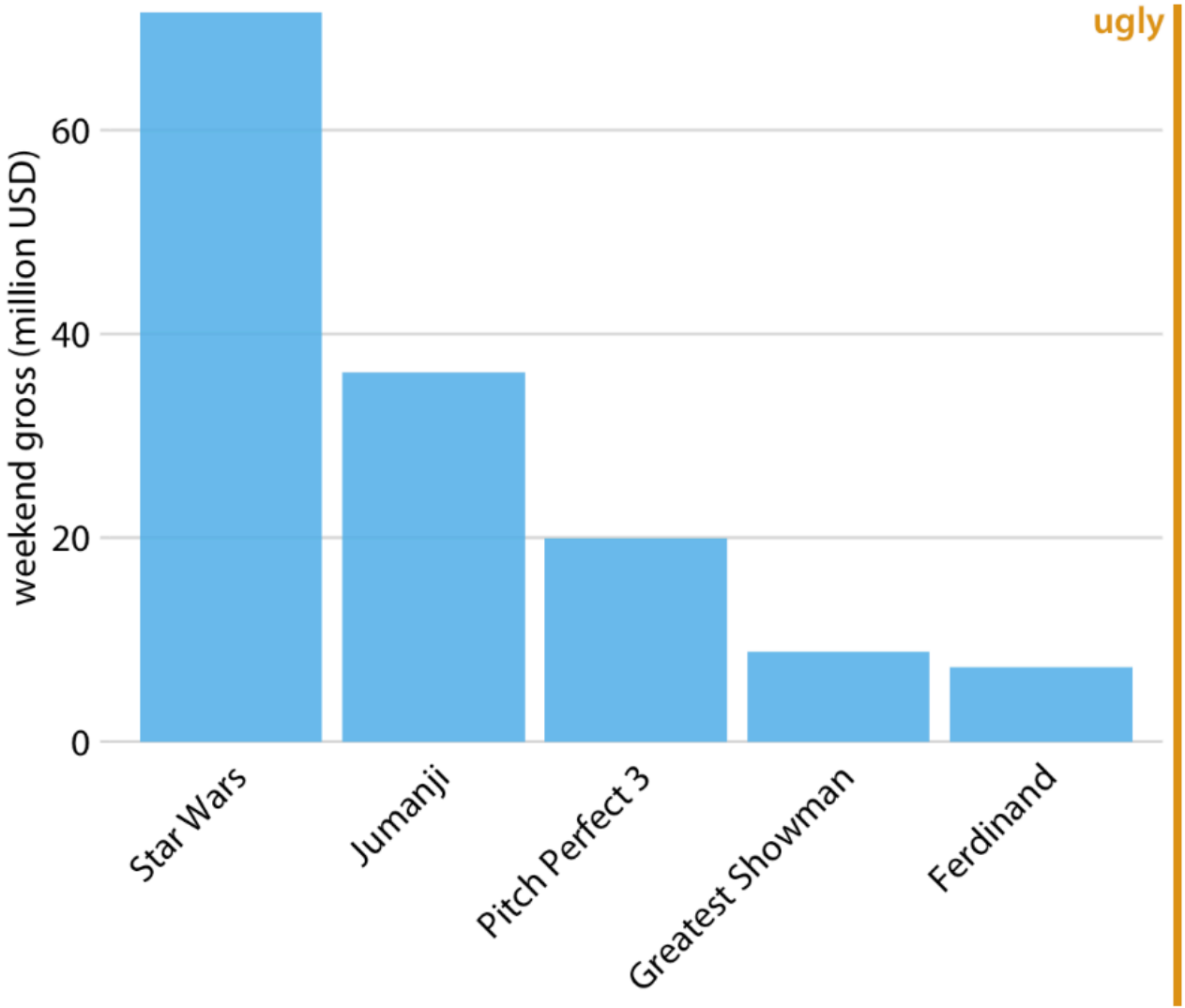

Candidates / (10.0 MeV)
10^4
10^3
10^2
5400
5500
5600
5700
5800
$m(\Lambda_c^+\overline{D}^0K^-)$ [MeV]
Unfiltered Data
Λ_c^+ mass
PID
$\overline{D}^0$ mass
BDT
IP Quality
Outlier
Lepton-hadron
Opening Angle
DTF Quality
Full Selection

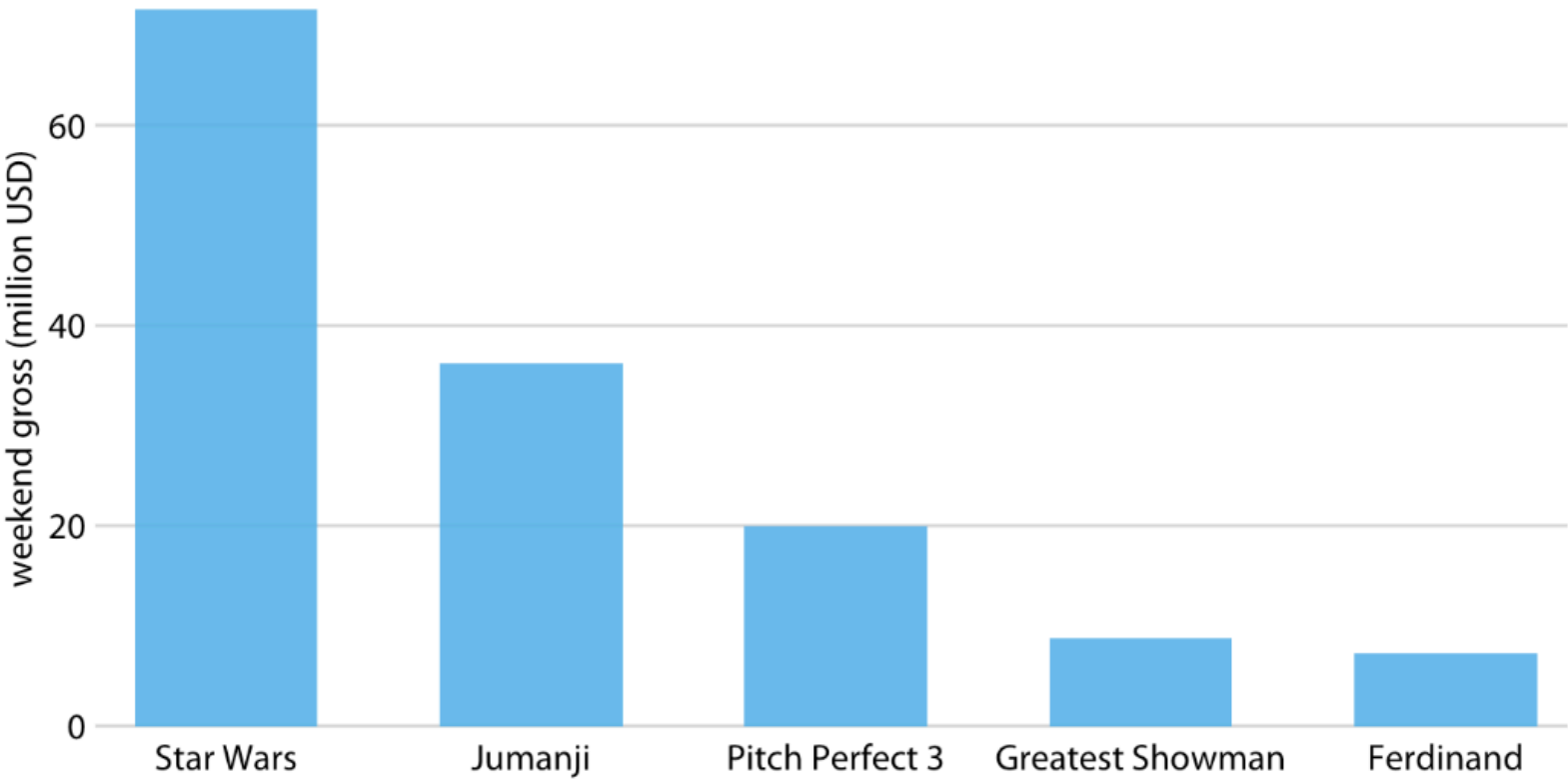

a
b
c
temperature (°F)
90
80
70
60
50
Jan
Apr
Jul
Oct
Jan
month

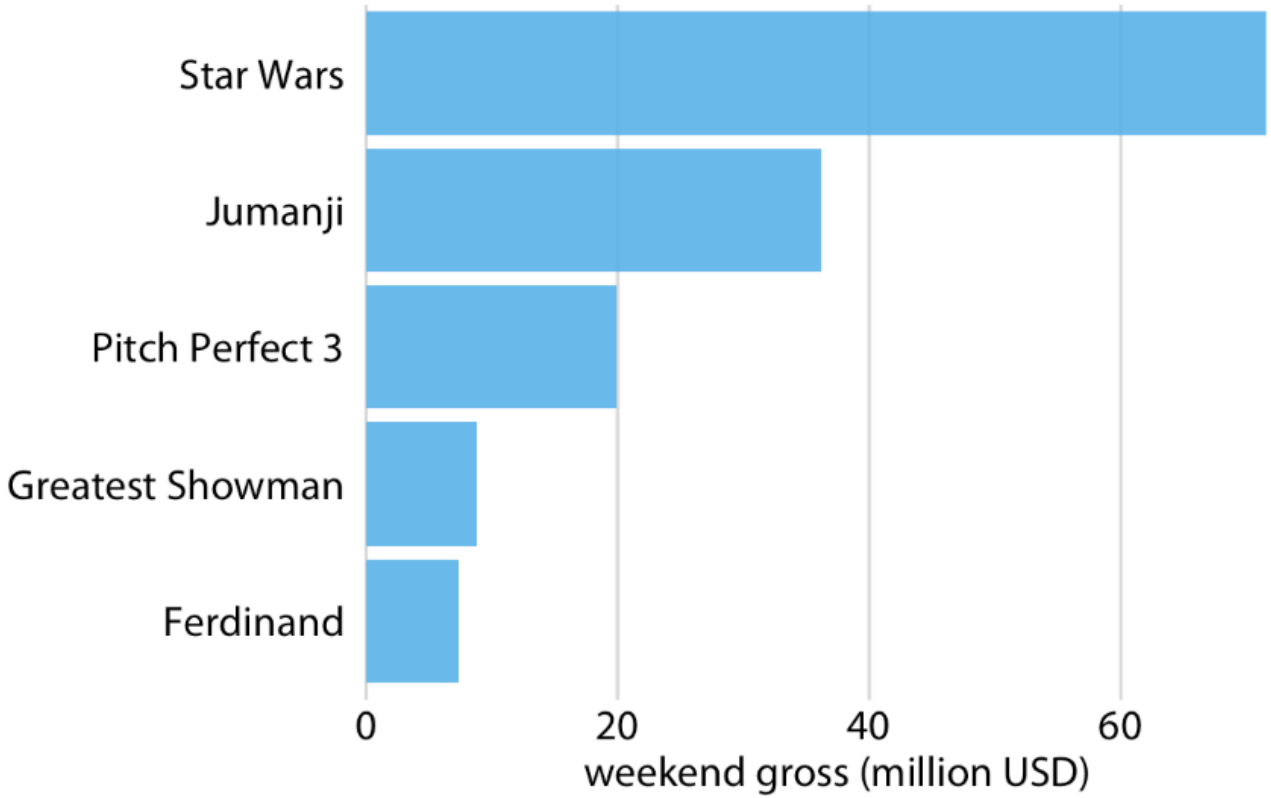

original data, linear scale
0
25
50
75
100
x
log-transformed data, linear scale
0.0
0.5
1.0
1.5
2.0
log10(x)
original data, logarithmic scale
1
3.16
10
31.6
100
x
logarithmic scale with incorrect axis title
wrong
1
3.16
10
31.6
100
log10(x)

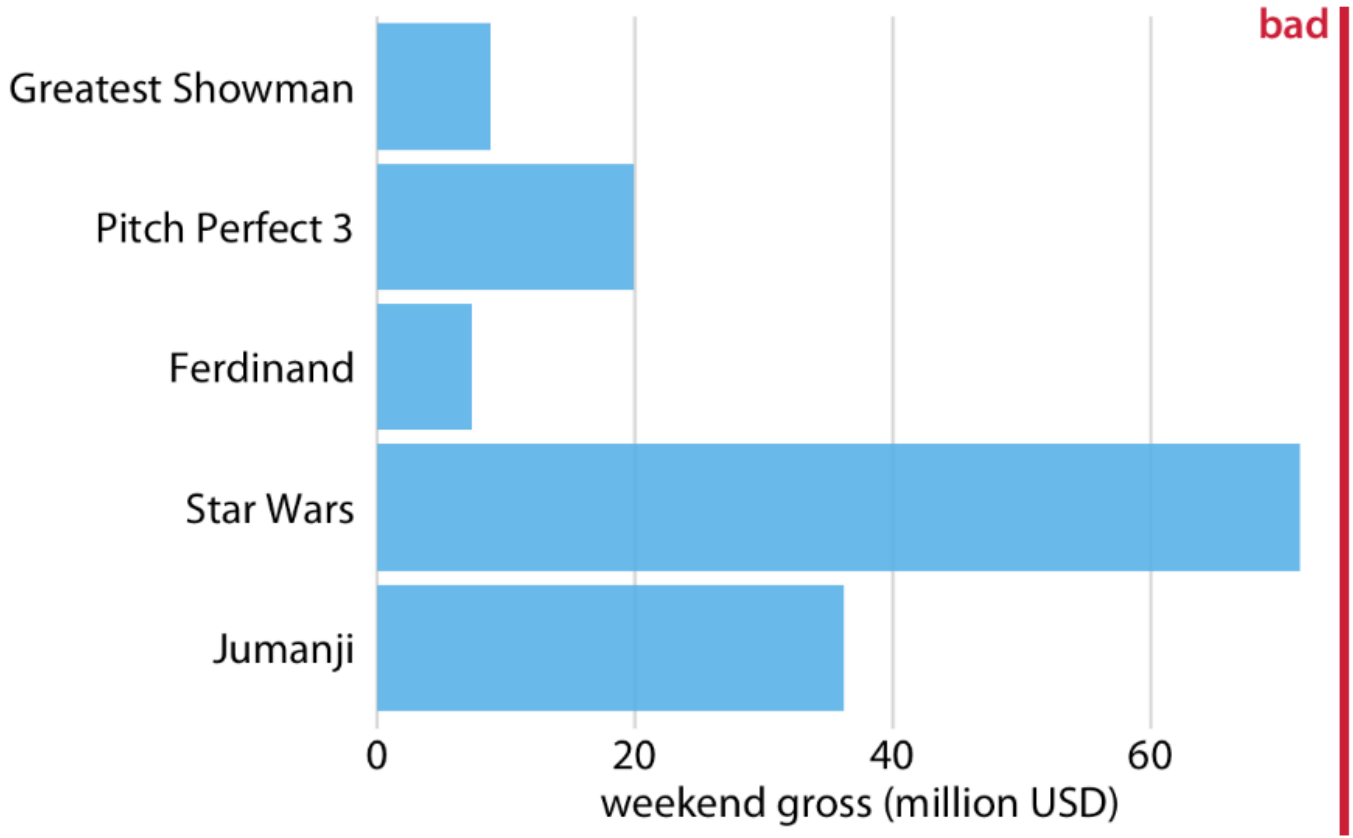
Visualizing **Amounts**

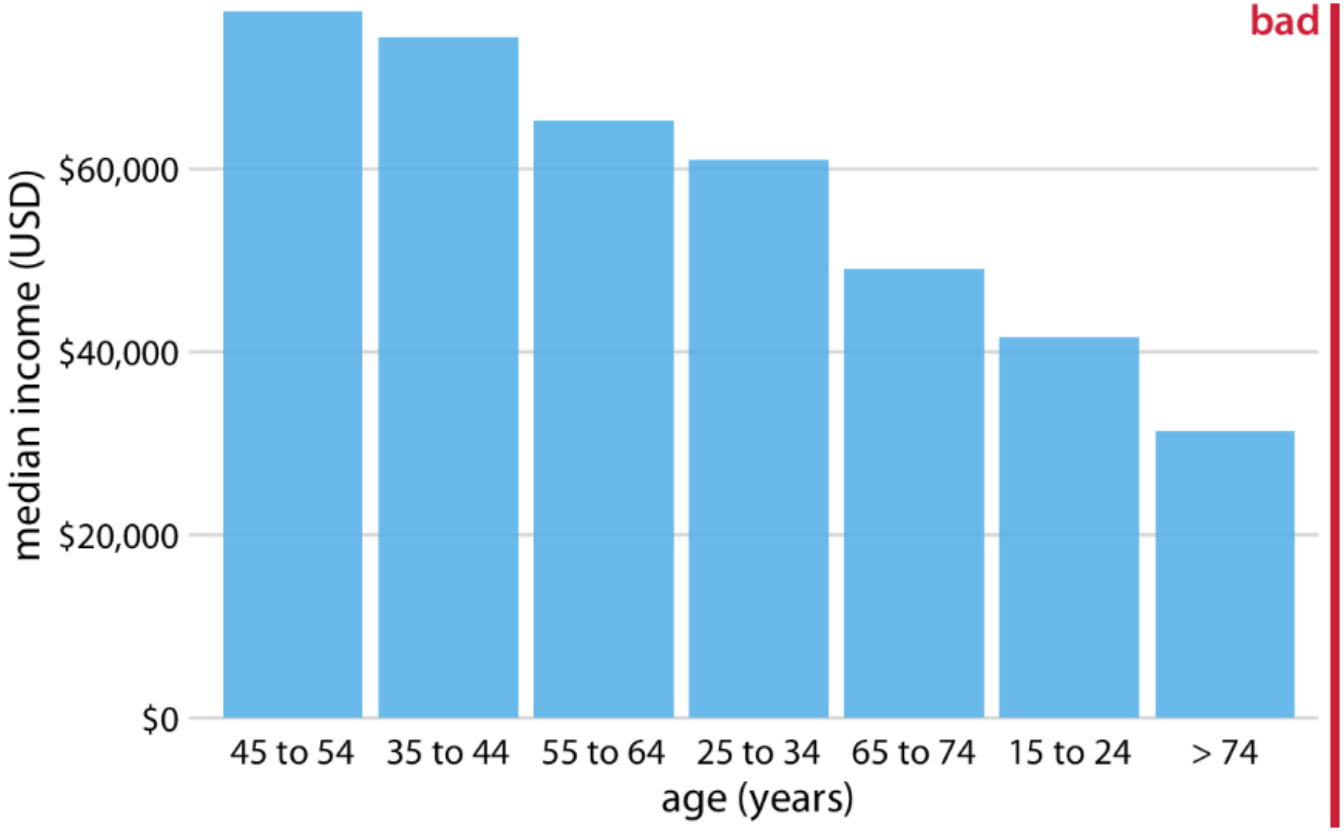
- Bar Charts
- Grouped Bar Charts
- Stacked Bar Charts
- Heat Maps

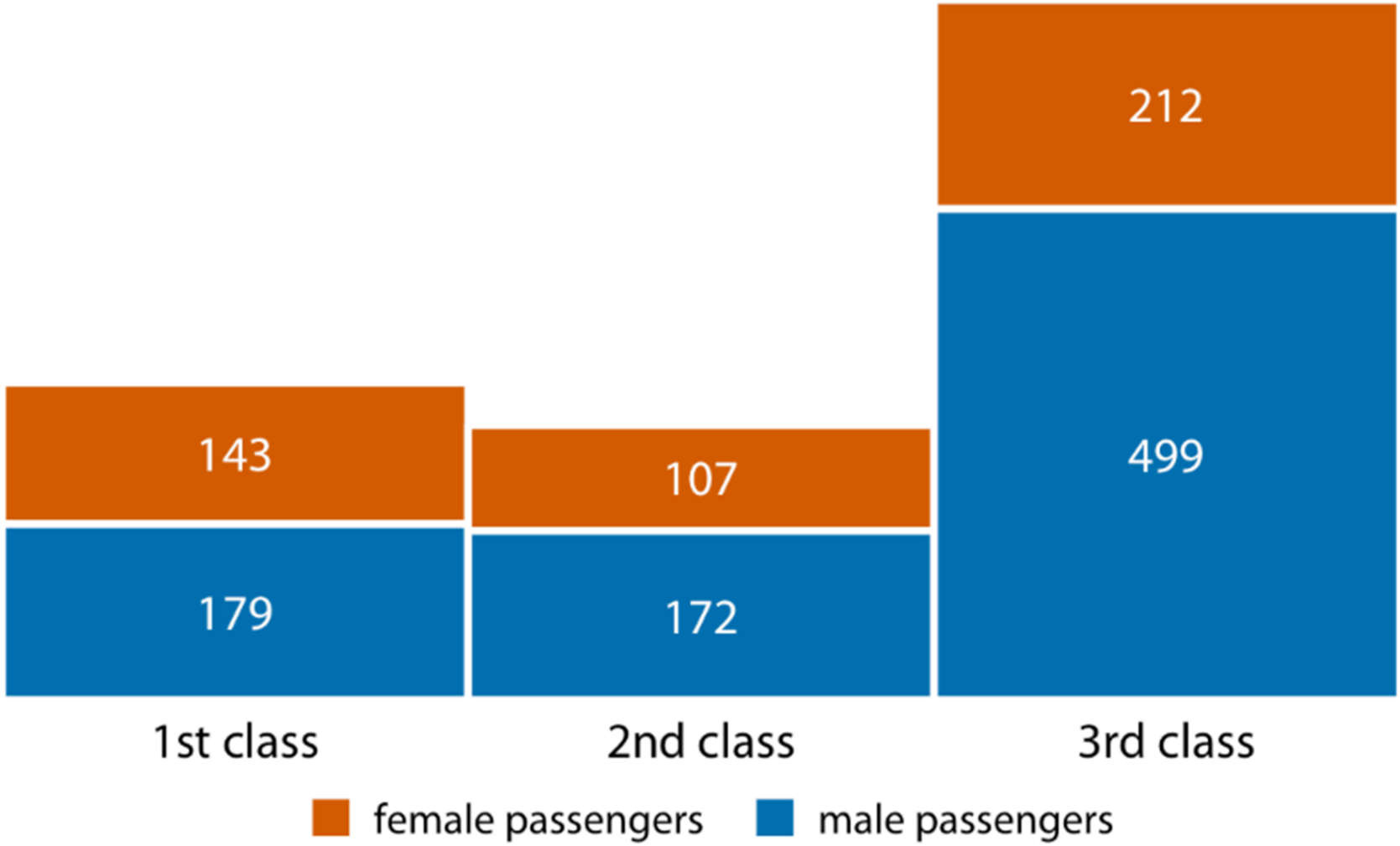

ugly
weekend gross (million USD)
60
40
20
0
Star Wars
Jumanji
Pitch Perfect 3
Greatest Showman
Ferdinand

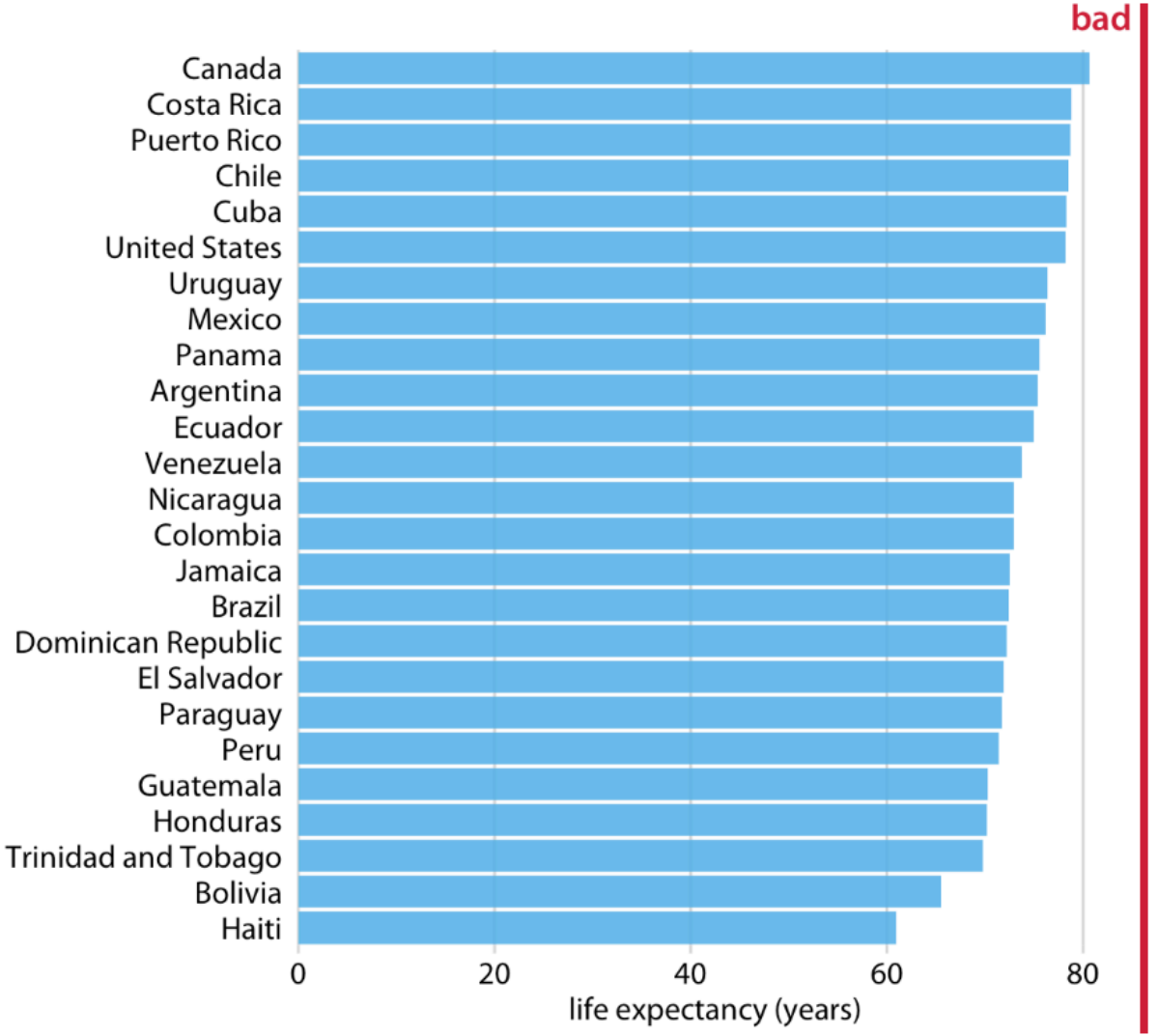

weekend gross (million USD)
60
40
20
0
Star Wars
Jumanji
Pitch Perfect 3
Greatest Showman
Ferdinand

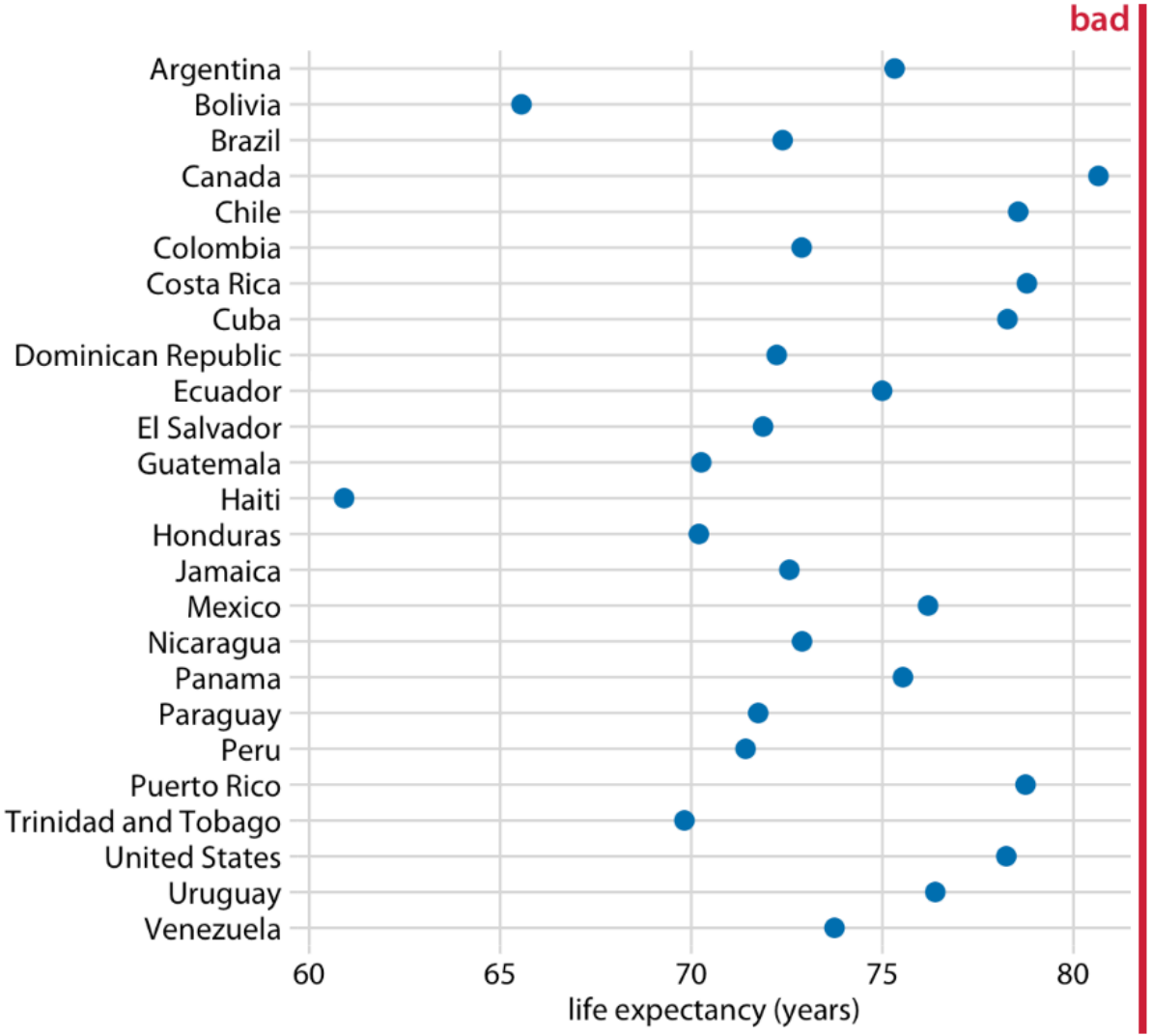

Star Wars
Jumanji
Pitch Perfect 3
Greatest Showman
Ferdinand
0
20
40
60
weekend gross (million USD)

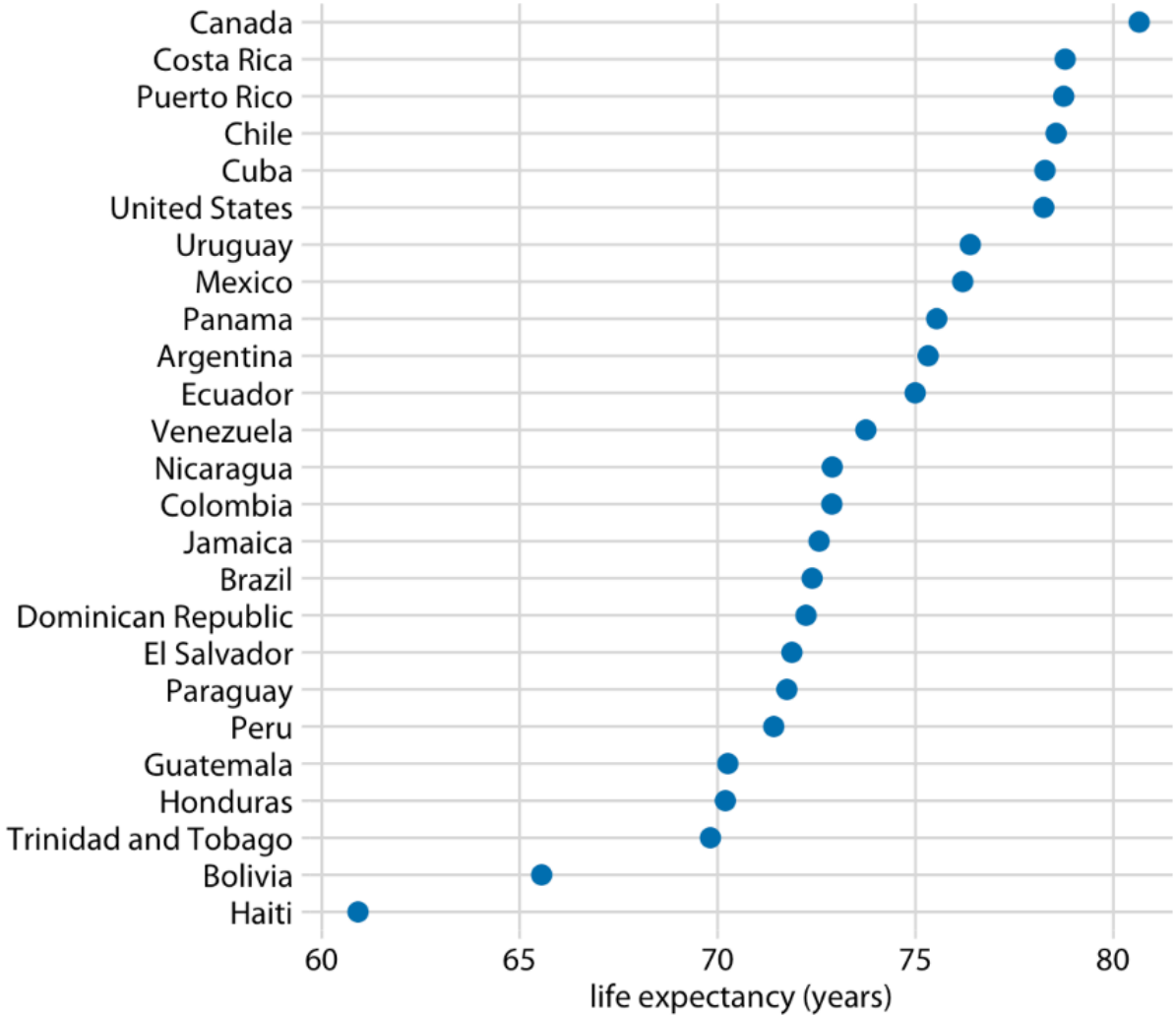

bad
Greatest Showman
Pitch Perfect 3
Ferdinand
Star Wars
Jumanji
0
20
40
60
weekend gross (million USD)

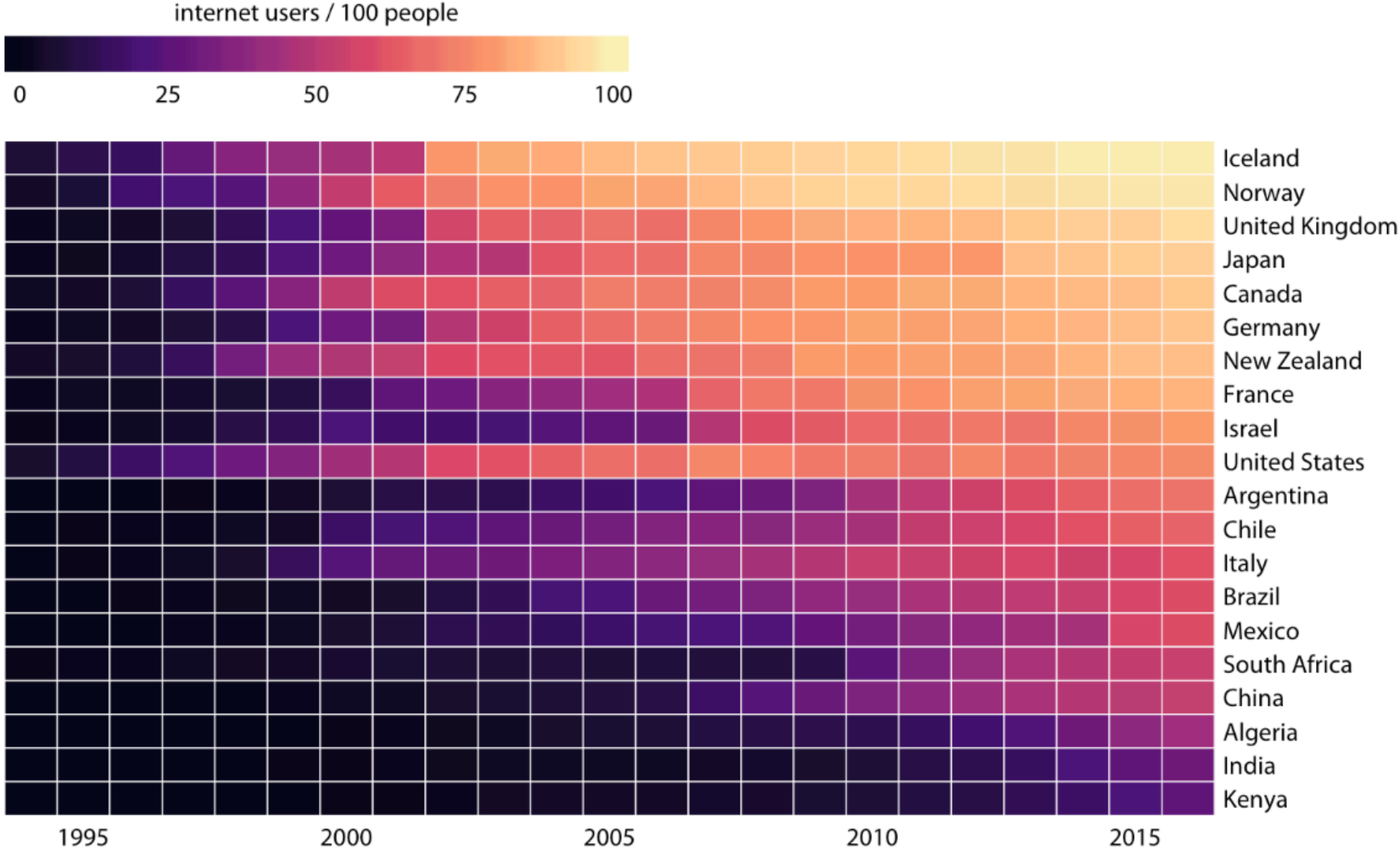

bad
median income (USD)
$60,000
$40,000
$20,000
$0
45 to 54
35 to 44
55 to 64
25 to 34
65 to 74
15 to 24
> 74
age (years)

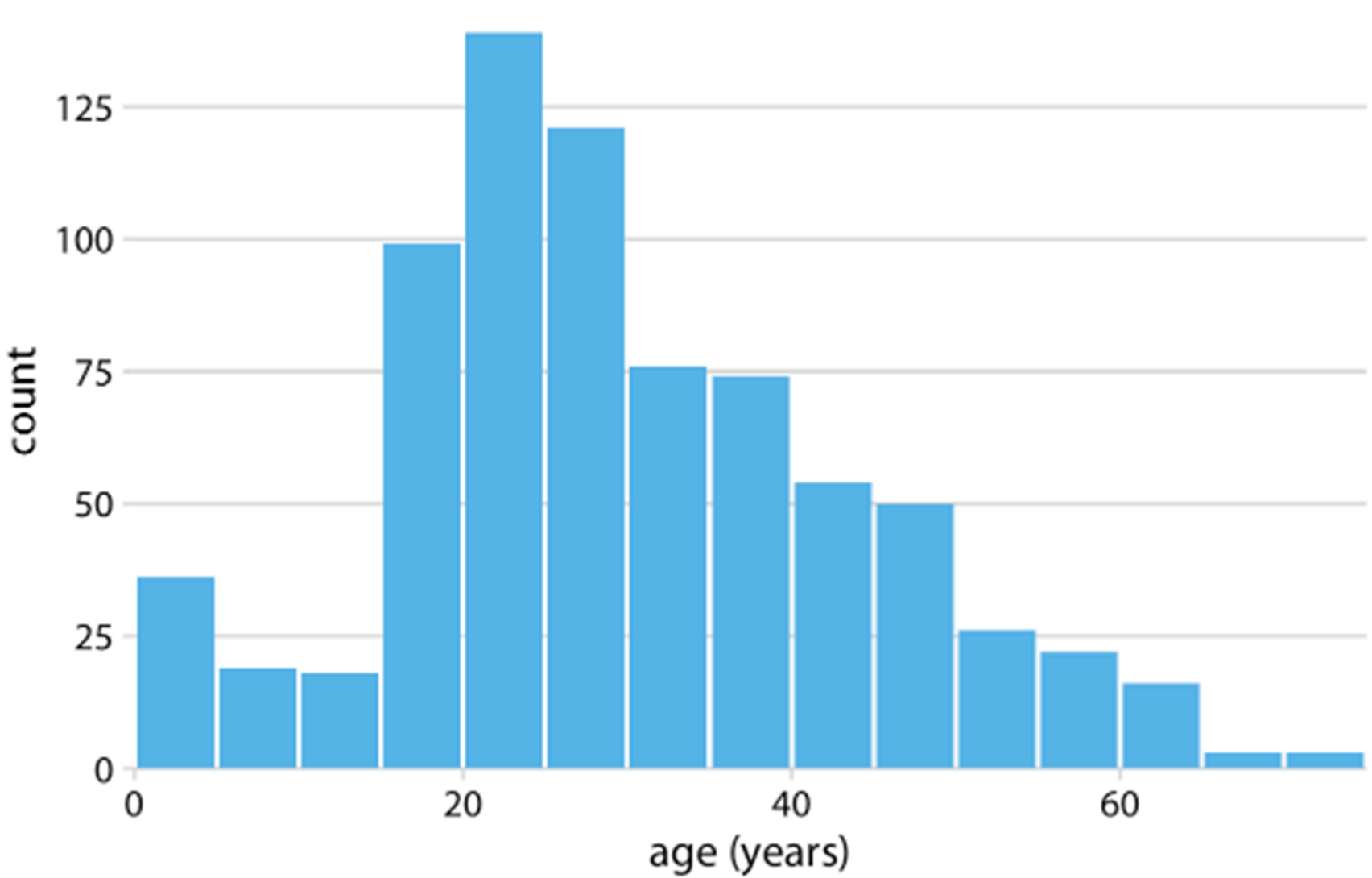

212
143
107
499
179
172
1st class
2nd class
3rd class
female passengers
male passengers

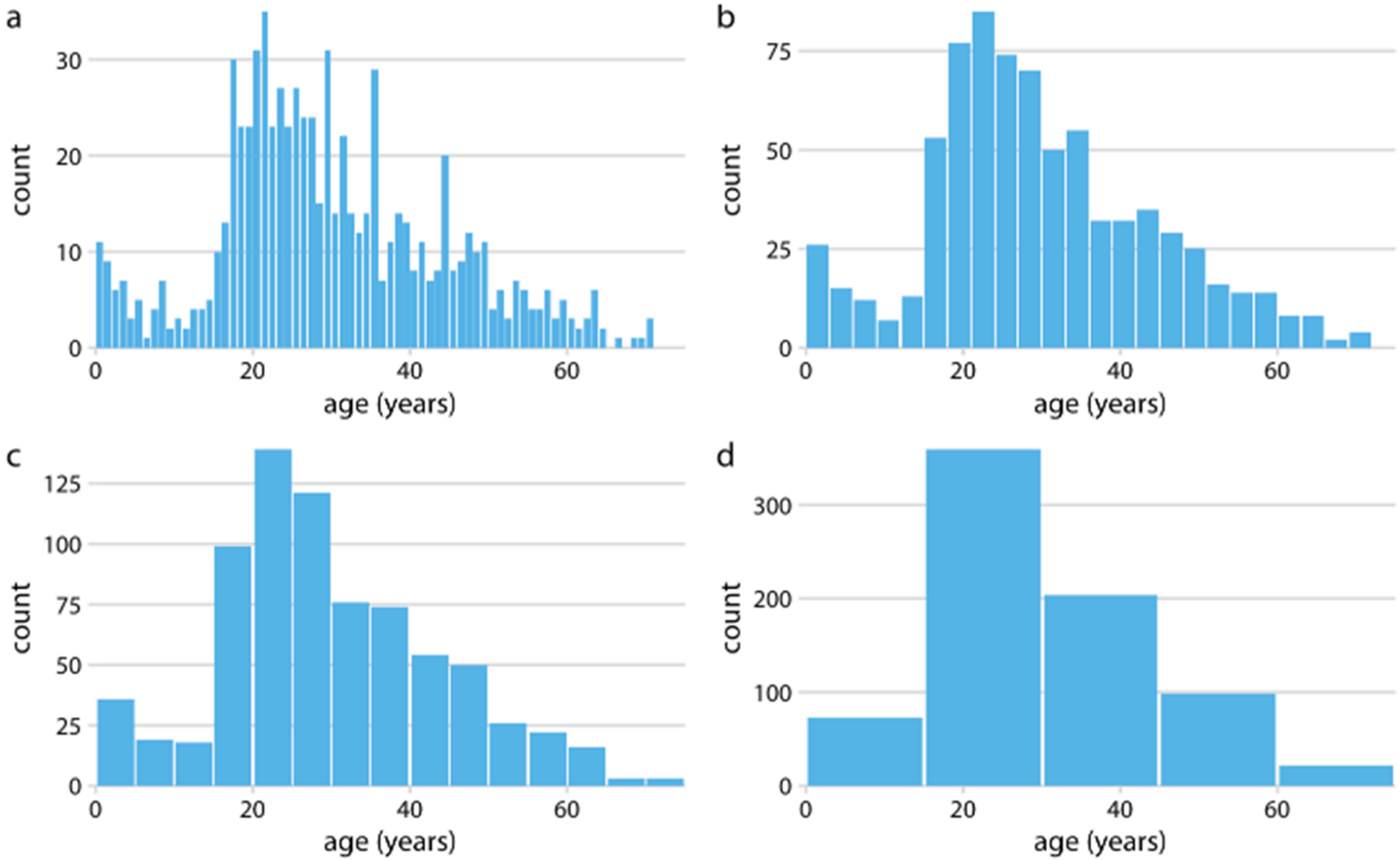

bad
Canada
Costa Rica
Puerto Rico
Chile
Cuba
United States
Uruguay
Mexico
Panama
Argentina
Ecuador
Venezuela
Nicaragua
Colombia
Jamaica
Brazil
Dominican Republic
El Salvador
Paraguay
Peru
Guatemala
Honduras
Trinidad and Tobago
Bolivia
Haiti
0
20
40
60
80
life expectancy (years)

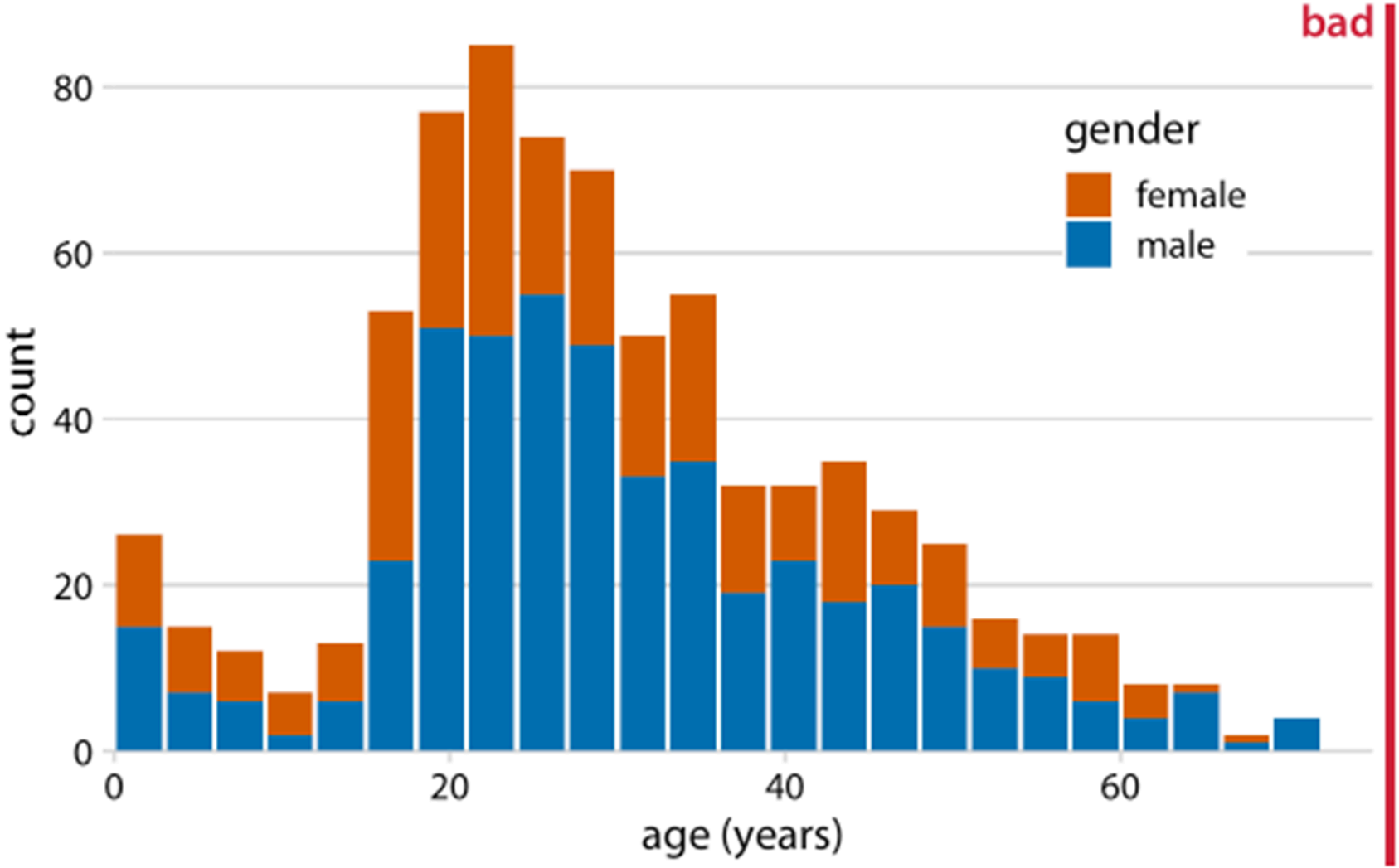

bad
Argentina
Bolivia
Brazil
Canada
Chile
Colombia
Costa Rica
Cuba
Dominican Republic
Ecuador
El Salvador
Guatemala
Haiti
Honduras
Jamaica
Mexico
Nicaragua
Panama
Paraguay
Peru
Puerto Rico
Trinidad and Tobago
United States
Uruguay
Venezuela
60
65
70
75
80
life expectancy (years)


Canada
Costa Rica
Puerto Rico
Chile
Cuba
United States
Uruguay
Mexico
Panama
Argentina
Ecuador
Venezuela
Nicaragua
Colombia
Jamaica
Brazil
Dominican Republic
El Salvador
Paraguay
Peru
Guatemala
Honduras
Trinidad and Tobago
Bolivia
Haiti
60
65
70
75
80
life expectancy (years)

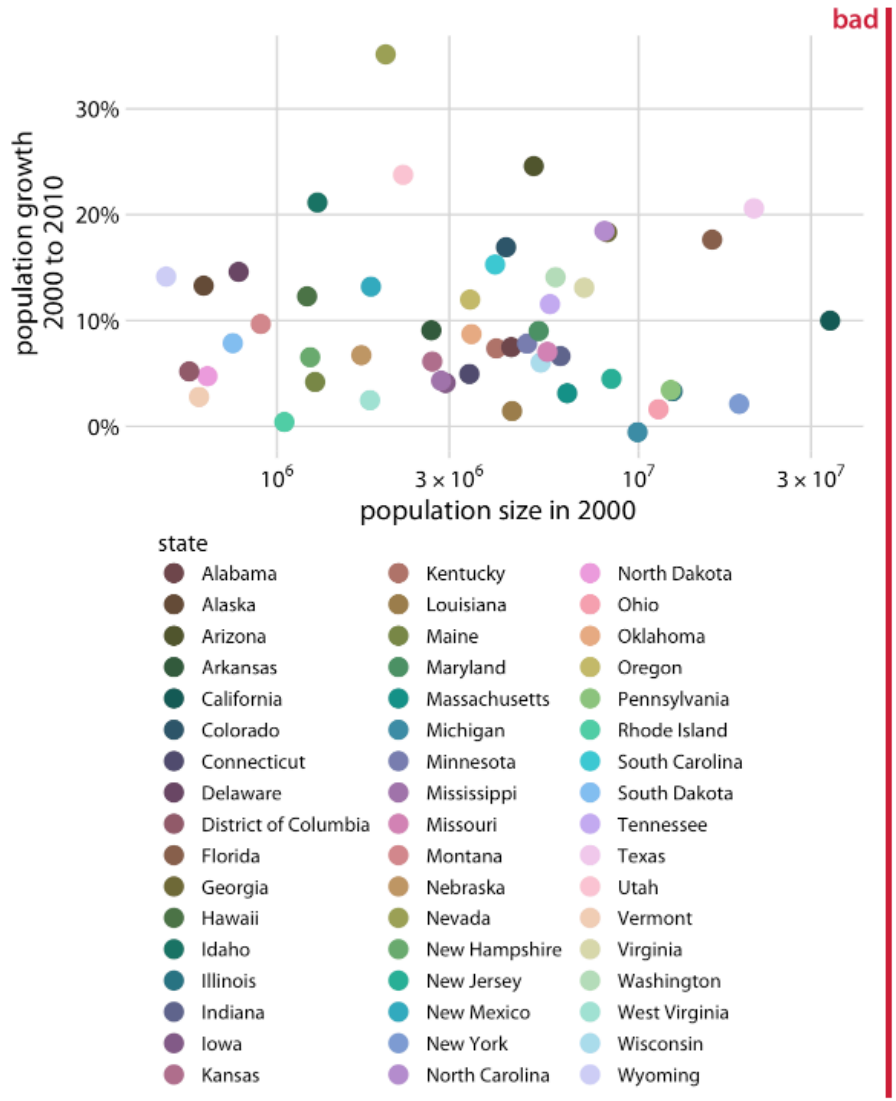

internet users / 100 people
0
25
50
75
100
Iceland
Norway
United Kingdom
Japan
Canada
Germany
New Zealand
France
Israel
United States
Argentina
Chile
Italy
Brazil
Mexico
South Africa
China
Algeria
India
Kenya
1995
2000
2005
2010
2015

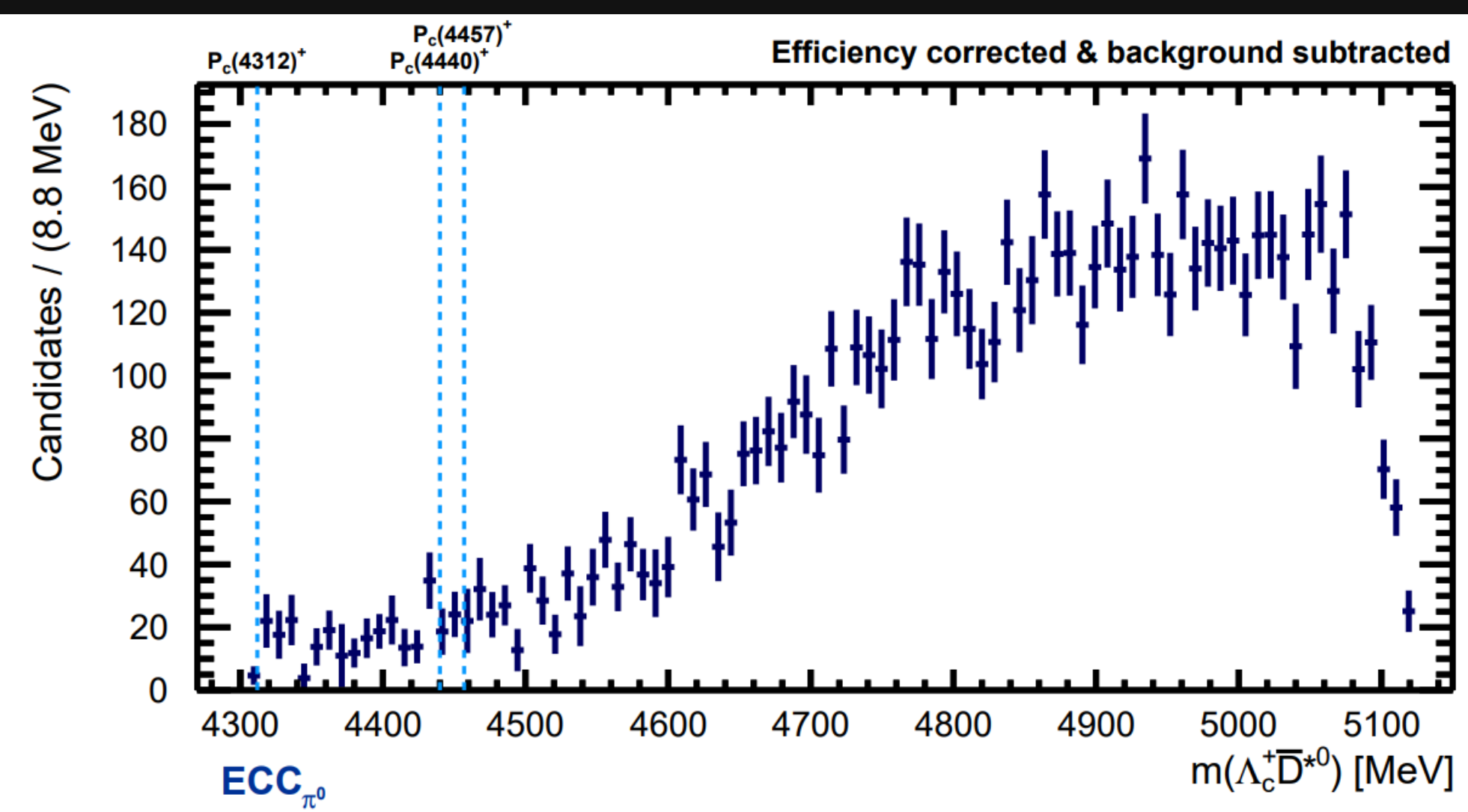

125
100
75
50
25
0
count
0
20
40
60
age (years)

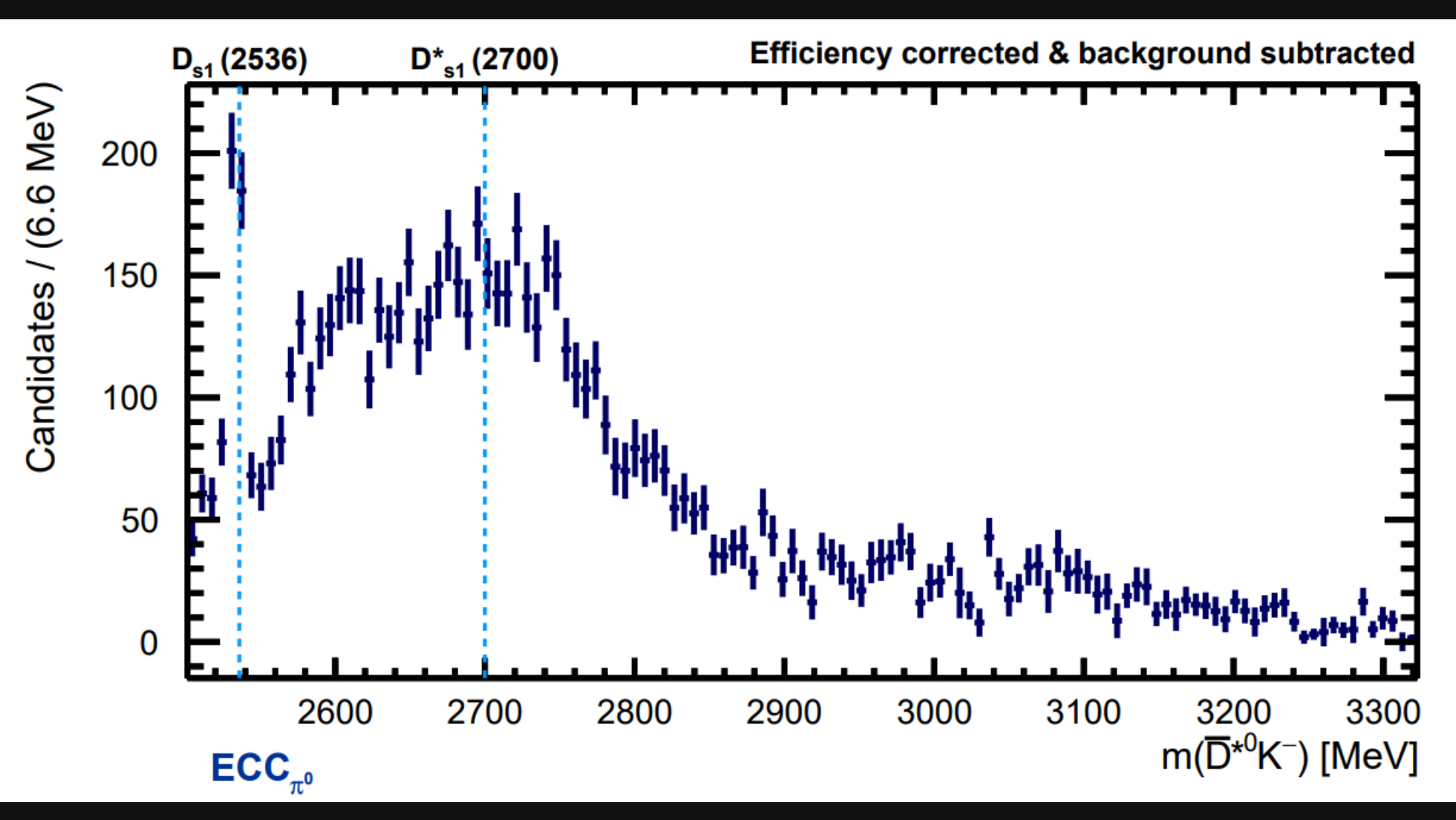

a
count
30
20
10
0
0
20
40
60
age (years)
b
count
75
50
25
0
0
20
40
60
age (years)
c
count
125
100
75
50
25
0
0
20
40
60
age (years)
d
count
300
200
100
0
0
20
40
60
age (years)


bad
gender
female
male
count
0
20
40
60
80
0
20
40
60
age (years)

bad


30%
20%
10%
0%
population growth
2000 to 2010
10^6
3×10^6
10^7
3×10^7
population size in 2000
state
Alabama
Alaska
Arizona
Arkansas
California
Colorado
Connecticut
Delaware
District of Columbia
Florida
Georgia
Hawaii
Idaho
Illinois
Indiana
Iowa
Kansas
Kentucky
Louisiana
Maine
Maryland
Massachusetts
Michigan
Minnesota
Mississippi
Missouri
Montana
Nebraska
Nevada
New Hampshire
New Jersey
New Mexico
New York
North Carolina
North Dakota
Ohio
Oklahoma
Oregon
Pennsylvania
Rhode Island
South Carolina
South Dakota
Tennessee
Texas
Utah
Vermont
Virginia
Washington
West Virginia
Wisconsin
Wyoming


$P_c(4312)^+$
$P_c(4457)^+$
$P_c(4440)^+$
Efficiency corrected & background subtracted
Candidates / (8.8 MeV)
0
20
40
60
80
100
120
140
160
180
4300
4400
4500
4600
4700
4800
4900
5000
5100
$m(\Lambda_c^+\overline{D}^{*0})$ [MeV]
ECC_{π^0}


$D_{s1}(2536)$
$D^*_{s1}(2700)$
Efficiency corrected & background subtracted
Candidates / (6.6 MeV)
200
150
100
50
0
2600
2700
2800
2900
3000
3100
3200
3300
$m(\bar{D}^{*0}K^-)$ [MeV]
ECC$_{\pi^0}$